Security-Preserving Distributed Samplers: How to Generate any CRS in One Round without Random Oracles

Damiano Abram[1], Brent Waters[2,3], and Mark Zhandry[3]

[1] Aarhus University
damiano.abram@cs.au.dk
[2] University of Texas at Austin
bwaters@cs.utexas.edu
[3] NTT Research
mzhandry@gmail.com

Abstract. A distributed sampler is a way for several mutually distrusting parties to non-interactively generate a common reference string (CRS) that all parties trust. Previous work constructs distributed samplers in the random oracle model, or in the standard model with very limited security guarantees. This is no accident, as standard model distributed samplers with full security were shown impossible.

In this work, we provide new definitions for distributed samplers which we show achieve meaningful security guarantees in the standard model. In particular, our notion implies that the hardness of a wide range of security games is preserved when the CRS is replaced with a distributed sampler. We also show how to realize our notion of distributed samplers. A core technical tool enabling our construction is a new notion of single-message zero knowledge.

1 Introduction

Many protocols require a common reference string to be generated by a third party in order to securely run the protocol. Importantly, the security of the protocol requires that the any secrets revealed during setup are hidden from the parties of the protocol. For example, if the protocol relies on a public RSA modulus for a reference string, the parties of the protocol must not know the prime factors. Such a structured common reference string requires placing enormous trust in the third party, and naturally leads to the question:

What happens if the trusted third party is actually not trustworthy?

Digging deeper, there may be many potential third parties who are willing to run the setup: maybe certain state organizations (e.g. NIST) as well and independent organizations (e.g. EFF). Some participants in the protocol may trust some third parties, while some participants only trust other third parties, and there may be no overlap between the trusted parties. How can we ensure that all protocol participants trust the reference string?

An obvious solution is for all potential third parties to run an MPC protocol to generate the reference string. Then, as long as each participant trusts a single third party, they will trust the reference string (CRS). However, engaging in an MPC protocol can be a logistical burden for these third parties. For comparison, in a situation where the CRS is generated by a single trusted third party, that party can simply post the reference string they produce to some public domain. In contrast, if many third parties are engaging in an MPC protocol to compute the reference string, this requires the many third parties to send several messages back-and-forth between each other.

Another issue is the difficulty of updating the CRS if we want to expand the number of involved trusted parties. For example, suppose third parties A, B, C engaged in an MPC protocol to generate a CRS such as an RSA modulus N. At some later date, users u, v wish to engage in a protocol using an RSA modulus, but user u only trusts a new third party D and not A, B, C. Meanwhile v does not trust D since it is new. Unfortunately, this would require A, B, C to come back online and interact with D to create a new modulus N'. A, B, C may be unable or unwilling to do so, as it would be an unreasonable burden to re-run the MPC any time a trusted setup was requested with a new third party.

Solution: Distributed Samplers. Abram, Scholl, and Yakoubov [ASY22] proposed the notion of a *distributed sampler*. Here, parties A, B, C each individually run their own setup algorithm locally, arriving at messages U_A, U_B, U_C, which they post to some public domain. Now when a set of users want a CRS generated by A, B, C, they look up U_A, U_B, U_C, and run a procedure which deterministically extracts a CRS from U_A, U_B, U_C. Because the process of computing the CRS from U_A, U_B, U_C is deterministic, all parties can compute it from U_A, U_B, U_C for themselves, and therefore do not require any additional interaction. Thus, the tuple U_A, U_B, U_C now acts as the common reference string, which is simply the concatenation of the individual messages of the various third parties. Informally, as long as a user trusts at least one of the third parties, then they trust the CRS derived from the list of strings that includes that party.

When a set of users wishes to incorporate a new third party D, all they need is for D to generate and post its own U_D. Now the parties can derive a new CRS from U_A, U_B, U_C, U_D. Importantly the original parties A, B, C do not need to do anything to add a new third party. In the follow-up work of [AOS23], a construction is given that maintains security in such a scenario.

Limitations of Existing Work. The work of [ASY22] constructs two kinds of distributed samplers both utilizing indistinguishability obfuscation. The first achieves semi-honest security, where the third parties *honestly* generate their messages but wish to then break a protocol using the generated CRS. Unfortunately, this notion of security is rather limited, since a truly malicious adversary could try to generate their messages dishonestly in order to influence the generated CRS. Such influence over the CRS offers much greater flexibility in breaking the protocol. For example, if the CRS is for a statistically sound proof system, a malicious adversary may try to influence the CRS into a “bad” one where false proofs exist.

The second distributed sampler achieves full malicious security in the UC model. However, the construction requires the random oracle model, and worse requires the full power of programming the random oracle.

Thus, the existing work either requires the full power of the random oracle model, or achieves only a very limited notion of security. This is no accident: as shown by [AOS23], full standard model malicious security is in fact *impossible.* So the question becomes: what kind of malicious security can be meaningfully achieved in the standard model?

1.1 Our Work

In this work, we address the above limitations of prior work, by giving new definitions for distributed samplers that avoid the above impossibility while still guaranteeing meaningful security against malicious adversaries, and providing a new instantiation of distributed samplers satisfying this definition. As a crucial step toward this goal, we also investigate single message zero knowledge proofs in the standard model, and provide new constructions with novel features. A summary of our main results follows.

Defining Distributed Samplers. Our first contribution is to define new security notions for distributed samplers. We describe a notion of *security preserving* distributed samplers, which implies that, for any game-based protocol using a reference string, security is preserved by the distributed sampler. That is, if the protocol is secure under a reference string generated by a single trusted third party, then it is also secure when the reference string is generated via a distributed sampler, as long as *at least one* of the parties involved is trusted. We also give some technical definitions of security for distributed samplers that are easier to reason about, and we show that these notions imply adequate notions of security preservation. See Sections 5 and 6 for details.

Constructing Distributed Samplers. Next, we show how to construct distributed samplers meeting our new definition. We obtain two flavours of the primitive: a CRS-less distributed sampler with security against uniform adversaries and a construction achieving security against non-uniform adversaries by relying on a short, reusable and unstructured CRS.

Our construction uses [ASY22] as a starting basis. However, we need to make several key changes. Critically, we face the following challenge: in order to justify that the reference string is “as good as” an honestly generated one, the reduction needs to be able to embed an actual honestly generated reference string N into the honest third party’s message, and somehow force the adversary to generate their own messages in a way that makes the derived reference string equal to N. But in the case of malicious adversaries, whatever strategy the reduction uses, the adversary can seemingly use as well to force the derived reference string to be their own, maliciously generated, N'.

Extractable 1-message zero knowledge. Resolving the above problem requires many tools. One of the main ones is a new 1-message zero knowledge proof, which crucially does not need a CRS. Now, such an object is normally considered impossible, but it can be possible if the simulator is allowed to be non-uniform while the adversary is required to be uniform. Such 1-message zero knowledge leveraging non-uniformity was considered before [BP04]. However, our use of zero knowledge requires several features, such as the ability for the reduction to extract the original proof from the sender's message, that were not present in existing 1-message zero knowledge. We therefore develop a new 1-message zero knowledge proof system with several useful features that we crucially leverage to achieve our notion of distributed samplers.

Updatability. The distributed samplers presented in this work assume that the set of participants is a-priori given. As a consequence, our constructions tolerate inactive parties (their distributed sampler messages can be generated using default randomness), but when new participants join, the protocol needs to restart.

Applications. A direct implication of our results is the existence of a 3-round OT protocol in the plain model (no CRS) with security against active, *uniform* adversaries and non-uniform simulation. This is achieved by directly applying our CRS-less distributed sampler to [PVW08]. More in general, our distributed samplers imply 3-round active MPC in the plain model (no CRS) with security against uniform adversaries and non-uniform simulation [BL18a].

Our distributed samplers can also be used to compile extractable NIZKs into 2-round zero-knowledge proofs of knowledge[4]. The resulting constructions either rely on a short, unstructured CRS or no CRS at all, depending on whether we aim for security against non-uniform adversaries or not. Furthermore, the 2-round protocols satisfy automatically concurrent security, independently of the properties of the original NIZKs.

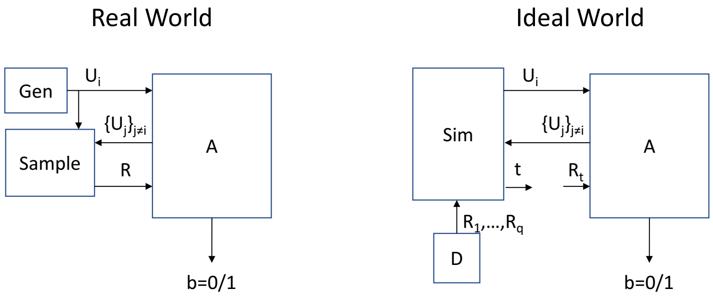
2 Technical Overview

2.1 New notions of distributed sampler

Full malicious security, and its impossibility. We first recall an informal description of the notion of malicious security obtained by [ASY22], which follows the real/ideal paradigm as shown in Figure 1 (We use $\mathcal{D}$ to denote the distribution of honestly generated CRSs. Such distribution can be private-coin). In the real world, the adversary is given the messages of the honest third parties, and then subsequently generates the messages of the malicious third parties. The challenger then derives the CRS from the combined messages of third parties, and gives it to the adversary. In the ideal world, the honest third party message is instead generated by a simulator (which depends on the adversary), and the

[4] Our techniques do not apply to non-extractable NIZKs. This is due to the challenger of the soundness game being not efficient.

simulator is given as input a CRS generated honestly from $\mathcal{D}$. The adversary is then given the simulated message and the honestly generated CRS. Security dictates that the two worlds are indistinguishable, which in particular implies that the derived CRS is equal to the provided honest CRS in the ideal world.



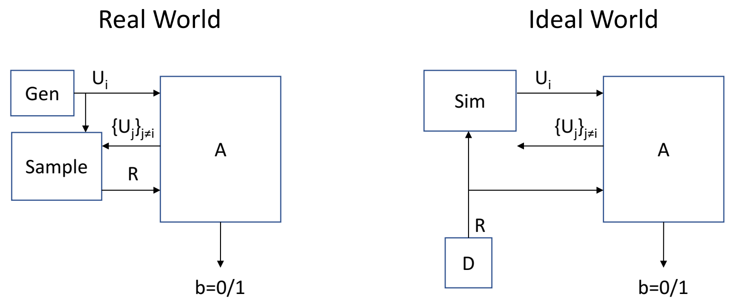
$$|\ \Pr[b = 1|\text{Real World}] - \Pr[b = 1|\text{Ideal World}]\ | < \mathsf{negl}$$

Fig. 1. An informal explanation of malicious security for distributed samplers. Here, Gen is the algorithm for honestly generating the third party messages U_j and Sample is the algorithm that combines the messages into the derived CRS R. i is the honest user, t is the simulator's choice of which of the honest CRS samples $R_1, \ldots, R_q$ to use.

This brief description is obviously impossible, however. Indeed, a malicious adversary could be *rushing*: *after* seeing the honest party's message, it could generate several sets of malicious third party messages (but even generate them honestly), compute the derived CRSs, and then select the set of third party messages that give a CRS most advantageous to the adversary. This means it is impossible for the simulator to guarantee that any single provided honest CRS is used by the adversary. To capture this ability of rushing adversaries, the definition actually gives the simulator a polynomial number of honestly generated potential CRSs, and the simulator can then choose which one gets sent to the adversary.

The above described notion of security is *still* impossible, as shown by [AOS23]. One basic reason is the following: the simulator has to produce a message U_i, whose length is fixed by the protocol. However, the sequence of honest CRSs provided to the simulator can be arbitrary long, since an arbitrary polynomial-time adversary can generate arbitrarily many sets of third party messages, thereby allowing them to select from an arbitrary polynomial number of CRSs. This means there is no way for a single U_i to embed all of the CRSs. [AOS23] formalize an impossibility, and it seems rather robust, since although their results apply only to the UC model with dishonest majority, different security settings such as standalone security, superpolynomial simulation, honest majority, or having the protocol depend itself on a CRS do not seem to solve the problem. The positive results of [ASY22,AOS23] therefore employ a random oracle. This avoids the impossibility, since the simulator can now program the random oracle with the various CRSs, instead of programming them into U_i. However, it requires the full power of programming the random oracle, and it is unclear what kind of security this gives in the standard model.

Our first notion: hardness-preserving distributed samplers. We now describe our new notions of security for distributed samplers. The first we describe is that of *hardness preserving*, which is given informally in Figure 2. There are two main differences from the security notion described. First, only a single honest CRS is given to the simulator in the ideal world. This is necessary in the standard model, as there is no way to program an unbounded number of CRSs into a fixed length simulated message. Note that with this change we can no longer hope to force the derived CRS to be equal to the provided honest CRS, except possibly with inverse polynomial probability. This means an adversary can distinguish real from ideal in the majority of cases. So the second change is to relax indistinguishability to the following. We only require that if the adversary outputs 1 in the real world with non-negligible probability ϵ_1, then it also outputs 1 in the ideal world with non-negligible probability ϵ_2. But ϵ_1 and ϵ_2 do not need to be close, and ϵ_2 can be far lower than ϵ_1.



$$\Pr[b = 1|\text{Ideal World}] < \mathsf{negl} \Rightarrow \Pr[b = 1|\text{Real World}] < \mathsf{negl}$$

Fig. 2. An informal explanation of hardness-preserving security for distributed samplers. It is the same as Figure 1, except that there is only a single honest CRS in the ideal world, and the relation between success probabilities in the two worlds is relaxed.

The obvious question is then: what kind of guarantees does such a relaxed definition provide? We show that hardness preserving distributed samplers are good for guaranteeing security for various *search* tasks. These are tasks where the adversary's goal is to output some value with non-negligible probability (as opposed to distinguishing tasks, where the goal is to output a value with probability non-negligibly larger than 1/2).

More precisely, we consider a general search game between a challenger and adversary, where at some step the challenger is provided with an honestly generated CRS, which it uses in its own internal logic but also sends to the adversary. We can compile such a game into one where the CRS is generated via distributed samplers, and the adversary controls all but one of the trusted third parties. A diagram of such a game and its compilation is given in Figure 3. We show the following:

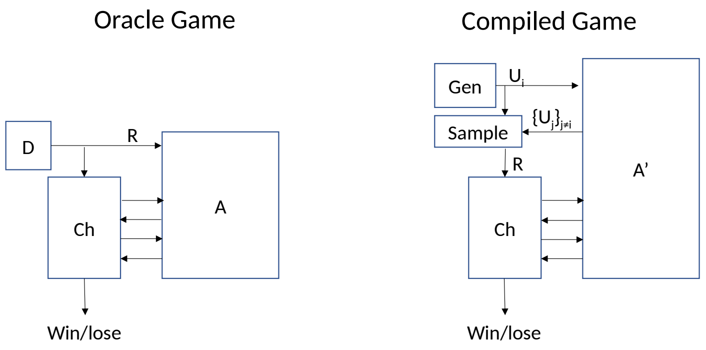


Fig. 3. Search games and their compilations. The figure on the left is a search game utilizing an honest CRS, while the figure on the right is the compiled game using a distributed sampler to generate the CRS.

Theorem 1 (informal). *If a distributed sampler is hardness-preserving and the search game is hard, then the compiled search game is also hard.*

Notice that there exists a non-negligible security loss between the original search game and the compiled version. Furthermore, the loss depends on the running time of the adversary. This is unavoidable: a rushing adversary can regenerate the corrupted party distributed sampler messages in its head many times, looking for an output that gives a higher chance of solving the search problem. The advantage will therefore degrade proportionally to the number of such trials, which is proportional to the running time.

Our second notion: indistinguishability-preserving distributed samplers. Hardness-preserving distributed samplers achieve a somewhat limited form of security against active adversaries. For starters, if the game is an indistinguishability game, the notion gives no guarantees. But a more subtle issue is the following. Consider a protocol like a NIZK with CRS. The definition of zero knowledge says that there exists a simulator which simulates both the CRS *and* the proof.

Perhaps it generates the CRS such that it knows a certain trapdoor, which allows it to generate a proof without knowing a witness. When using a distributed sampler, we would like the ideal world to reflect this simulated CRS and proof. But this is not a simple matter of plugging in the existing simulated CRS into the simulator for the distributed sampler, as there is no way for the distributed sampler simulator to then use the CRS trapdoor to help generate the proof. In the language of protocols and functionalities, this means that for a protocol Π with CRS which implements a functionality $\mathcal{F}$, the compiled protocol Π' using the distributed sampler to generate the CRS might no longer implement $\mathcal{F}$.

The second distributed sampler notion we introduce, called *indistinguishability-preserving*, tries to tackle this problem. The concept is informally described in Fig. 4: an indistinguishability-preserving distributed sampler compiles any protocol Π with CRS satisfying the condition at the top of Fig. 4 for some functionality $\mathcal{F}$ and simulator Sim_{Π}, into a protocol without CRS satisfying the property at the bottom.

We focus for a moment on the property at the top of Fig. 4. The condition states that the protocol Π implements the functionality $\mathcal{F}$. However, it actually gives a strictly stronger requirement: in the ideal world, the CRS is simulated using a distribution $\mathcal{D}'$ that produces both a sample R and a trapdoor T. While the adversary receives only R, the simulator Sim_{Π} receives also T. In the NIZK example, $\mathcal{D}'$ would be the trapdoored CRS, and T is the trapdoor. An important point is that the simulated CRS is independent of any information known to the functionality. Not all protocols have this kind of simulation. For example, the HSS construction of [OSY21] satisfies the property: the CRS is a large RSA modulus distributed identically to the protocol and simulated before interacting with the functionality. On the other hand, imagine a protocol where the CRS consists of an RSA modulus N. Suppose that the protocol allows, e.g., generic MPC modulo N and N is chosen by the functionality (notice that the CRS is given by the functionality). If we use an indistinguishability-preserving distributed sampler to generate N, the compiled protocol will not implement the functionality anymore. This is because, in the simulation, we cannot ensure that the output of the distributed sampler is the modulus N chosen by the functionality.

Moving on to the bottom of Fig. 4, we observe that, in the ideal world, the sampling algorithm of the distributed sampler has been substituted with a new algorithm called $\mathsf{Trapdoor}$. The latter has the purpose of extracting the trapdoors from the outputs of the simulated distributed sampler. The resulting values are then given to Sim_{Π}. Observe that the property at the bottom implies that the compiled protocol implements $\mathcal{F}$.

Theorem 2 (informal). *If a distributed sampler is indistinguishability-preserving and the protocol Π implements the functionality $\mathcal{F}$ as in the top of Fig. 4, then the compiled protocol also implements $\mathcal{F}$.*

The definition of indistinguishability-preserving distributed sampler is actually more general than what we outlined here: it provides security guarantees even when the sample from $\mathcal{D}$ is not given as a CRS but as an "oracle sample"

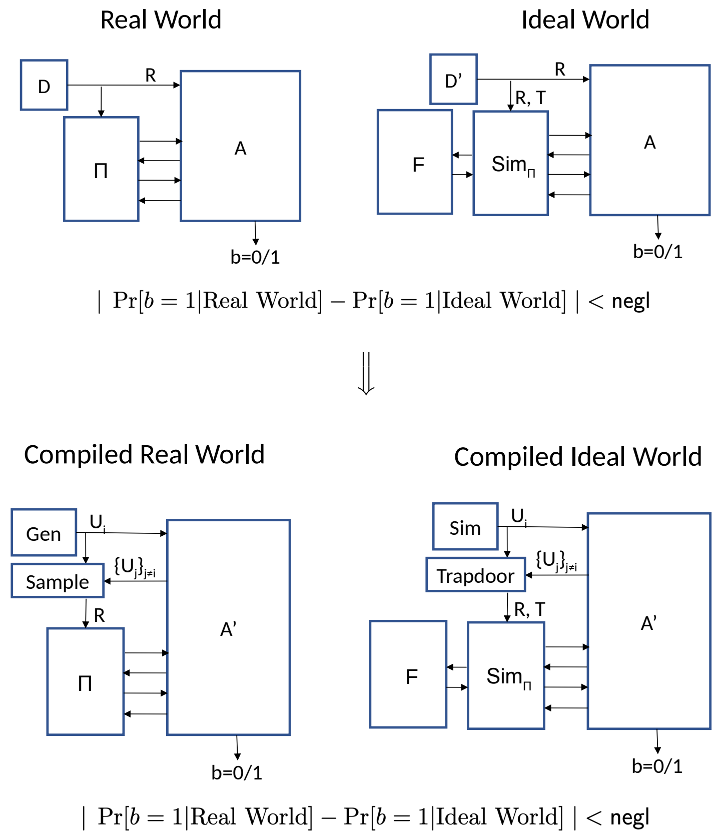


Fig. 4. An informal explanation of indistinguishability-preserving security for distributed samplers.

revealed halfway through the execution of the protocol. It is still possible to compile this kind of protocol using a distributed sampler: instead of executing it at the beginning, the parties will run it at a later stage. Sometimes, when the first round of the protocol Π is independent of the CRS, this fact allows us to compile Π without adding rounds of interaction. For more details, check Section 5.2.

Lossy distributed samplers. In the paper, we introduce one last notion: *lossy distributed samplers.* This will be a convenient technical notion that will help us realize our notions of distributed samplers from above. Such a lossy sampler consists of two modes of operation: in addition to a standard mode, in which the output remains unpredictable as long as one party is honest, there exists a lossy mode. When the latter is activated, the output becomes predictable: with overwhelming probability, it will lie in a set of polynomial size determined by the messages of the honest parties. Distinguishing between standard and lossy mode will always be possible, however, for any given PPT adversary. But by choosing sufficiently large parameters for the lossy mode, we ask that the distinguishability advantage for any given adversary can be made an arbitrarily small non-negligible function, i.e. for every PPT $\mathcal{A}$ and $\delta = 1/\mathsf{poly}$, there exists a sufficiently large q[5] such that

$$\Big|\Pr\big[\mathcal{A} \to 1 \big| \mathsf{StandardMode}\big] - \Pr\big[\mathcal{A} \to 1 \big| \mathsf{LossyMode}(q)\big]\Big| \leq \delta.$$

From lossy to hardness-preserving distributed samplers. We use lossy distributed samplers to build hardness-preserving distributed samplers. Consider an adversary $\mathcal{A}$ that, in the real-world game of the hardness-preserving distributed samplers (see Fig. 2), interacts with the standard mode of the construction. The idea is that if the adversary outputs 1 with non-negligible probability, we can activate the lossy mode with sufficiently large parameters so that $\mathcal{A}$ keeps outputting 1 with non-negligible probability. The main difference is that, now, the output of the construction is all of a sudden predictable.

At this point, we make use of a property that is satisfied by some lossy distributed samplers: *programmability.* The latter guarantees that we can hide an ideal sample $\hat{R} \xleftarrow{\$} \mathcal{D}$ among the outputs of a lossy-mode distributed samplers without the adversary's realizing. Since the output space is polynomial in size, the adversary ends up obliviously selecting $\hat{R}$ as output of the protocol with $p = 1/\mathsf{poly}$ probability. Conditioned on this event, $\mathcal{A}$ still outputs 1 with non-negligible probability ϵ. In conclusion, in the ideal world, the challenger just needs to send lossy-mode messages. The adversary will output 1 with probability at least $p \cdot \epsilon$.

Theorem 3 (Informal). *Any programmable, lossy distributed sampler is hardness-preserving.*

2.2 Building lossy distributed samplers

We explain how to build programmable, lossy distributed samplers using, among other tools, indistinguishability obfuscation [GGH$^+$13], multi-key FHE [AJJM20], extremely lossy functions (ELFs) [Zha16] and a new primitive called *almost everywhere extractable NIZKs.* We make extensive use of subexponentially secure primitives. The resulting lossy distributed sampler makes use of a short (polynomial in λ, but independent of $\mathcal{D}$), unstructured and reusable CRS (the construction is secure even if the CRS is reused in multiple concurrent instantiations of the protocol, potentially involving different subsets of parties). Our construction originates from the semi-honest distributed sampler of [ASY22]. We briefly recall it.

[5] q is a polynomial that upper bounds the size of the output space.

The encryption program. In [ASY22], a distributed sampler message consists of two obfuscated programs. Adapting the terminology to this paper, we call them the *encryption program* and the *decryption program*.

The encryption program of party P_i takes care of generating a multi-key FHE encryption of a random string s_i under a fresh key pk_i. The output of the construction will be obtained by adding the n random strings $s_1, \ldots, s_n$ and feeding the result as randomness for $\mathcal{D}$, i.e. the output sample is $R := \mathcal{D}(\mathbb{1}^n; s_1 \oplus \cdots \oplus s_n)$. Observe that thanks to the homomorphic properties of multi-key FHE, given the encryptions of the random strings, everybody is able to derive an encryption of R[6]. The issue is that nobody is able to decrypt it: the output of the multi-key FHE evaluation is encrypted under a "joint key". In order to decrypt, the parties usually need to collaborate: each of them performs a partial decryption of the joint ciphertext and publishes the result. By pooling together the partial plaintexts, everybody can reconstruct the hidden message.

The decryption program. Usually, a multi-key FHE decryption requires interaction. In the distributed samplers of [ASY22], however, the decryption program takes care of everything without needing additional rounds of interaction.

Formally, the decryption program of party P_i takes as input the encryption programs of all the parties and evaluates them. After receiving the encryption of s_j for every $j \in [n]$, the program retrieves an encryption of the output R by applying homomorphic operations on the ciphertexts. Observe that all the decryption programs derive the same joint ciphertext C. The execution terminates by performing a partial decryption of C using the private counterpart of pk_i. The program outputs the resulting partial plaintext.

Observe that by evaluating all the decryption programs, the parties are able to retrieve all the partial decryptions of C. At that point, reconstructing R is immediate.

Counteracting the residual function attack. A common issue of all 1-round MPC protocols is that an adversary can rerun the protocol in its head many times changing a subset of the messages. The outputs of all these executions are correlated with the inputs of the honest parties. For particular functionalities, this could leak enough information to reconstruct the input of the honest parties.

In distributed samplers, there are no private inputs but we still need to be careful: we need to make sure that, in every distributed sampler execution, the encryption programs use independent looking random strings $s_1, \ldots, s_n$. If that was not the case, the adversary might use the residual function attack to learn information about the randomness used in the main execution of the protocol.

In [ASY22], the authors ensure this by feeding the encryption program of each party with the hash of the encryption programs of the other players (notice that inputting the program themselves would not be possible for a matter of sizes). The encryption program generates the randomness for the multi-key FHE

[6] The fact that the ciphertexts are encrypted under different keys does not constitute a problem.

```
$\mathsf{EProg}[K_i]$

Hard-coded. The PPRF key $K_i$.
Input. A digest $y$.

$(s_i, r_i, r'_i) \leftarrow F(K_i, y)$
$(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{mkFHE.Gen}(1\!\!1^\lambda;\, r_i)$
$c_i \leftarrow \mathsf{mkFHE.Enc}(\mathsf{pk}_i, s_i;\, r'_i)$
Output $(\mathsf{pk}_i, c_i)$.
```

Fig. 5. A sketch of the unobfuscated encryption program of party P_i

```
$\mathsf{DProg}[K_i, \mathsf{EP}_i, \sigma, (\mathsf{id}_j)_{j \neq i}]$

Hard-coded. The PPRF key $K_i$, the encryption program $\mathsf{EP}_i$, the CRS for a NIZK $\sigma$, the identities of the other parties $(\mathsf{id}_j)_{j \neq i}$.
Input. Set of $n-1$ tuples $(\mathsf{EP}_j, \pi_j)_{j \neq i}$ where $\mathsf{EP}_j$ is the encryption program of party $P_j$ and $\pi_j$ is a NIZK proving its well-formedness.

$\forall j \neq i: \quad b_j \leftarrow \mathsf{NIZK.Verify}(\sigma, \mathsf{id}_j, \pi_j, \mathsf{EP}_j)$
If $\exists j \neq i$ such that $b_j = 0$, output $\perp$
$\forall j \in [n]: \quad y_j \leftarrow \mathsf{Hash}\big((\mathsf{EP}_l)_{l \neq j}\big)$
$\forall j \in [n]: \quad (\mathsf{pk}_j, c_j) \leftarrow \mathsf{EP}_j(y_j)$
$C \leftarrow \mathsf{mkFHE.Eval}\big(\mathcal{D}, c_1, \ldots, c_n\big)$
$(s_i, r_i, r'_i) \leftarrow F(K_i, y_i)$
$(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{mkFHE.Gen}(1\!\!1^\lambda;\, r_i)$
$d_i \leftarrow \mathsf{mkFHE.PartDec}(C, \mathsf{sk}_i)$
Output $d_i$
```

Fig. 6. A sketch of the unobfuscated decryption program of party P_i

key pk_i and the string s_i by inputting the hash into a puncturable PRF. Observe that if any adversary reruns the distributed sampler in its head modifying any of the other messages, the hash fed in the encryption program changes. As a consequence, the program will use an independent looking s_i (and an independent looking multi-key FHE key pair).

In our lossy distributed sampler, the encryption program will remain the same as in [ASY22]. We sketch its code in Fig. 5.

Adding extractable NIZKs. The main change we bring to the construction is to add non-interactive zero knowledge (NIZK) proofs of the well-formedness of the encryption programs. These proofs will be inputted into the decryption programs. When any of the proofs do not verify, the decryption program will output $\perp$. We sketch their code in Fig. 6.

```
$\mathsf{EProg}_{\mathsf{Ls}}[K_i]$

Hard-coded. The PPRF key $K_i$.
Input. A digest $y$.

1. $(\eta_i, \eta'_i) \leftarrow F'(K_i, y)$
2. $(\phi, \mathsf{pk}_i, c_i) \leftarrow \mathsf{mkFHE.Sim}_1(1\!\!1^\lambda;\ \eta_i)$
3. Output $(\mathsf{pk}_i, c_i)$.
```

Fig. 7. A sketch of the unobfuscated encryption program for the lossy mode

In order to describe the lossy mode of the distributed sampler, we assume that the NIZK is *extractable*, which means there is a special trapdoor that allows for extracting from any proof the witness used to generate the proof. We defer the discussion of the exact properties needed until later in this overview.

The lossy mode of the distributed sampler tweaks the programs of one of the honest parties as follows. The encryption program will generate simulated public keys and ciphertexts. The decryption program, instead of verifying the NIZKs, will extract the witnesses from them using the extraction property of the NIZK. From the latter, it will derive the randomness used to generate the multi-keys FHE keys and ciphertexts of the other players. At that point, similarly to [HIJ$^+$17], it simulates the partial decryption instead of directly performing it. We recall that the simulator for the partial decryption takes as input a targeted plaintext R' [AJJM20]. Such value might differ for the actual message hidden in the joint ciphertext C, however, the output of the decryption is still guaranteed to be R'.

Decreasing the size of the output space using an ELF. In the lossy mode, the output of the protocol is decided by the party that sends the lossy-mode programs (those that simulate the multi-key FHE operations). How can we restrict the output space to a set of polynomial size without the adversary's immediate detecting the small output space? After all, the adversary could keep generating outputs, hoping to find a collision. After only a polynomial number of outputs, the adversary would expect to find such a collision in the lossy mode.

To rectify this issue, we have the size of the lossy output space be a polynomial that grows with the adversary's run time and success probability, making sure it is a sufficiently large polynomial that the adversary cannot detect it in the time give.

At a lower level, we use extremely lossy functions (ELFs) [Zha16]. These are randomized algorithms generating deterministic functions with large domain. The primitive has two modes of operations: injective mode and lossy mode. When the first mode is activated, the function is injective. In the other case, the image of the function has size smaller than q, where q is a polynomial parameterizing the lossy mode. The two modes will be always distinguishable with non-negligible advantage. ELFs guarantee that, for any adversary $\mathcal{A}$ and inverse-

$\mathsf{DProg}_{\mathsf{Ls}}[K_i, \mathsf{EP}_i, \sigma, (\tau_e^j)_{j \neq i}, K, f]$

Hard-coded. The PPRF key K_i, the encryption program EP_i, the CRS for the almost everywhere extractable NIZK σ, the extraction trapdoors $(\tau_e^j)_{j \neq i}$, a PPRF key K, an ELF f.
Input. Set of $n-1$ tuples $(\mathsf{EP}_j, \pi_j)_{j \neq i}$ where EP_j is the encryption program of party P_j and π_j is an almost everywhere extractable NIZK proving its well-formedness.

```
1. $\forall j \neq i: \quad K_j \leftarrow \mathsf{NIZK.Extract}(\tau_e^j, \pi_j, \mathsf{EP}_j)$
2. If $\exists j \neq i$ such that $K_j = \bot$, output $\bot$
3. $\forall j \in [n]: \quad y_j \leftarrow \mathsf{Hash}\big((\mathsf{EP}_l)_{l \neq j}\big)$
4. $\forall j \neq i: \quad (s_j, r_j, r'_j) \leftarrow F(K_j, y_j)$
5. $z \leftarrow f(\mathsf{EP}_1, \ldots, \mathsf{EP}_n)$
6. $s \leftarrow F(K, z)$
7. $R' \leftarrow \mathcal{D}(1\!\!1^\lambda; s)$
8. $(\eta_i, \eta'_i) \leftarrow F'(K_i, y_i)$
9. $(\phi, \mathsf{pk}_i, c_i) \leftarrow \mathsf{mkFHE.Sim}_1(1\!\!1^\lambda; \eta_i)$
10. $d_i \leftarrow \mathsf{mkFHE.Sim}_2\Big(\phi, \mathcal{D}, R', (s_j, r_j, r'_j)_{j \neq i}; \eta'_i\Big)$
11. Output $d_i$
```

Fig. 8. A sketch of the unobfuscated decryption program for the lossy mode

polynomial δ, by choosing a sufficiently large polynomial q, it is possible to make the distinguishability advantage between the injective mode and the lossy mode smaller than δ.

In our construction, we generate the value R' input in the partial decryption simulator by applying an ELF on the concatenation of the encryption programs of the n parties. The result is then fed in a puncturable PRF. Its output is used as randomness for $\mathcal{D}(1\!\!1^\lambda)$. In this way, when the ELF has a small image, the distributed sampler will have a small output space. We skecth the code of the lossy-mode programs in Fig. 7 and Fig. 8.

Programmability. It is easy to see that our candidate distributed sampler is programmable: in order to hide an ideal sample $\hat{R}$ in the output space, we can just pick a random value $\hat{z}$ in the image of the ELF f and input $\hat{R}$ into the partial decryption simulator whenever $f(\mathsf{EP}_1, \ldots, \mathsf{EP}_n) = \hat{z}$. By the security of puncturable PRFs, the changes cannot be detected by the adversary. Furthermore, if the ELF satisfies an additional property called *regularity* [Zha16], it is guaranteed that the event $f(\mathsf{EP}_1, \ldots, \mathsf{EP}_n) = \hat{z}$ occurs with inverse-polynomial probability.

2.3 Security Proof Challenge 1: Simultaneous Extraction and Statistical Soundness

At this point, we can try to prove the security of the candidate lossy distributed sampler. However, there are some challenges that need to be overcome.

The first challenge is the following. In the lossy mode, we need to be able to extract witnesses from valid proofs. However, zero knowledge implies that there are false proofs that contain no witnesses. The existence of these false proofs presents a problem for proving security using indistinguishability obfuscation.

More generally, consider the following general setup. There is a program C_0 receiving n values $x_1, \ldots, x_n$ as inputs from n parties along with n NIZKs proving their validity. The program C_0 outputs $\perp$ whenever any of the NIZKs does not verify. In the other cases, it outputs $C(x_1, \ldots, x_n)$ where C is some circuit. There also a second program C_1 that, instead of verifying the NIZKs, it tries to extract the witnesses hidden in them (C_1 outputs $\perp$ if the extraction of any witness fails). Then it uses the extracted witnesses to attempt to simulate the same behavior as C_0. The goal is to have obfuscations of C_0 and C_1 be indistinguishable.

The problem of differing inputs. The main issue is that C_0 and C_1 have differing inputs: the zero-knowledge property of the NIZKs guarantees the existence of proofs for which the witness cannot be extracted despite verification succeeds. On these inputs, the behavior of C_0 and C_1 can be easily told apart. In order to apply indistinguishability obfuscation, however, we need C_0 and C_1 to be equivalent programs.

Fortunately, finding these differing inputs is hard. Therefore the natural tool to achieve indistinguishability between obfuscations of C_0 and C_1 would be differing-input obfuscation [BGI+01]. The existence of such primitive is, however, controversial due to some results suggesting its impossibility [GGHW14,BSW16]. In [HIJ+17], Halevi *et al.* faced a problem similar to ours. They solved it by designing NIZKs that can be simulated only for statements hidden in the CRS. Since there is a small number of problematic statements, it is easy to take care of the corresponding executions of C_0 and C_1 using just indistinguishability obfuscation. The solution of Halevi *et al.*, however, compromises the reusability of the CRS and makes it grow with the size of the statements. Since we want to keep the CRS as simple as possible, we follow a different approach.

Indistinguishability obfuscation is enough. We rely solely on indistinguishability obfuscation. In [BCP14], Boyle, Chung and Pass showed that, if two programs have a polynomial number of differing inputs and finding any of them is hard, then iO is enough to hide which program was obfuscated. In our application, the number of differing inputs is of course superpolynomial, however, we notice that the result of [BCP14] can be generalized: assume that all differing inputs have a prefix in a set S. If finding an element in S is hard even for adversaries running in time $\mathsf{poly}\big(\lambda, |S|\big)$, subexponentially secure iO is sufficient to hide which program was obfuscated.

To leverage this observation, we introduce the notion of *almost everywhere extractable NIZKs*. Such NIZKs are designed so that the prefix of all the valid proofs for which the witness cannot be extracted lies in a set S. Finding an element in S is hard even for adversaries running in time $\mathsf{poly}\big(\lambda, |S|\big)$ that are provided with the extraction trapdoor. By using almost everywhere extractable NIZKs together with the generalization of [BCP14], we can show that P_0 and P_1 are hard to distinguish despite the existence of differing-inputs. We discuss building such NIZKs later in this overview.

2.4 Security Proof Challenge 2: More Differing Inputs

Decreasing the entropy of the encryption programs. At this point, we can try to prove the security of the candidate lossy distributed sampler. The strategy is the following: using the properties of the almost everywhere extractable NIZKs followed by an input-by-input iO argument, we show that, if the ELF is in injective mode, the lossy-mode programs are indistinguishable from the usual ones. By switching to a lossy ELF, we can then argue that the distinguishability advantage between the modes of the distributed sampler can be made an arbitrarily small inverse-polynomial function.

There is only one problem that hinders this plan: beyond the differing-inputs caused by the NIZK extraction (which are taken care by the almost everywhere extractable NIZKs), there exist other inputs for which the lossy-mode programs have a clearly distinguishable behaviour. Consider indeed two tuples of encryption programs $(\mathsf{EP}_j)_{j\neq i}$ and $(\mathsf{EP}'_j)_{j\neq i}$ having colliding hashes. When these tuples are used along with normal programs for party P_i, the outputs of the protocol will be correlated: in both executions, the programs of P_i use the same random string s_i (see how s_i is generated in Fig. 5). If instead P_i sent lossy-mode programs, the outputs will look independent of each other (see how $\hat{R}$ is generated in Fig. 8).

Even if these problematic inputs are hard to find, this time we do not use the trick by Boyle, Chung and Pass [BCP14]. To work around the issue, we decrease the entropy of the encryption programs: we require that they are generated using the randomness produced by a PRG with a small λ-bit seed. The almost everywhere extractable NIZKs will guarantee that the adversary does not break this rule. On the other hand, the lossy-mode programs will use full-entropy randomness. In this way, the total number of valid encryption programs for the corrupted parties becomes smaller than $(2^\lambda)^{n-1}$. By adopting a subexponentially collision-resistant hash function, we can make sure that, with overwhelming probability, there exist no collisions among these $(2^\lambda)^{n-1}$ elements. Moreover, the digests will still be small enough to fit into the encryption programs.

This technique solves also circular dependencies between subexponentially secure primitives: the input-by-input iO argument requires us to work with a number of hybrids that is proportional to the number of valid encryption programs. In each of these hybrids, we need to rely on the security of multi-key FHE. In order for the proof to go through, the size of the multi-key FHE keys therefore needs to increase logarithmically with the number of hybrids. If we

used full-entropy encryption programs, the size of the keys would be so large that they would not even fit in the encryption programs anymore. By forcing valid encryption programs to have low entropy, we can hybrid over only the valid programs instead of all possible encryption programs, thereby eliminating the circular dependency. The properties of the NIZK guarantee not only that the adversary cannot find non-valid encryption programs, but that they do not even exist.
With these challenges overcome, we prove the following:

Theorem 4 (Informal). *Assuming almost everywhere extractable NIZKs, subexponential iO, subexponential multi-key FHE, subexponentially collision-resistant hash functions and regular extremely lossy functions, the distributed sampler sketched above is lossy and programmable.*

2.5 Building indistinguishability-preserving distributed samplers.

A lossy distributed sampler is not necessarily indistinguishability-preserving. We show, however, that the construction described above actually is:

Theorem 5 (Informal). *Assuming almost everywhere extractable NIZKs, subexponential iO, subexponential multi-key FHE, subexponentially collision-resistant hash functions and regular extremely lossy functions, the distributed sampler sketched above is indistinguishability-preserving.*

We start by considering a protocol Π that relies on a CRS sampled from the distribution $\mathcal{D}$. We suppose that Π implements a functionality $\mathcal{F}$ as described at the top of Fig. 4. In particular, in the ideal world, the CRS is simulated using a distribution $\mathcal{D}'$ that outputs a trapdoor T along with the sample R.

A sketch of the proof. We use a hybrid argument beginning from the compilation of the real world using the standard mode of our lossy distributed sampler and ending with the compilation of the ideal world using a simulated mode (see the bottom of Fig. 4). We prove that the compiled worlds are computationally indistinguishable.

As a first step, we switch the distributed sampler to lossy mode. This already introduces some non-negligible distinguishability advantage in the proof, we will explain later why this does not constitute a problem. On the other hand, the lossy mode allows us to move to a sample space of polynomial size.

Next, we gradually change the distribution of the outputs of the distributed sampler, switching from $\mathcal{D}$ to $\mathcal{D}'$. The technique here is rather simple: we just rely on the security of puncturable PRFs similarly to what we did to argue programmability. Along the way, we gradually switch from the execution of Π, to the execution of the simulator Sim_{Π}. In particular, there will some subhybrids in which some of the distributed sampler outputs come from $\mathcal{D}$ and some from $\mathcal{D}'$. We run Sim_{Π} only when the adversary chooses an execution where the sample comes from $\mathcal{D}'$. In these cases, we can retrieve the trapdoor by using the puncturable PRF key K and the ELF hidden in the lossy-mode programs (see Fig. 8).

Observe that, since the sample space is small, switching from $\mathcal{D}$ to $\mathcal{D}'$ needs only a polynomial number of subhybrids. As a consequence, we do not need that $\mathcal{D}$ and $\mathcal{D}'$ are subexponentially indistinguishable, nor that Π implements $\mathcal{F}$ with subexponential security.

In the last hybrid, we switch the ELF in the lossy-mode programs back to injective mode. Once again, the operation introduces a non-negligible distinguishability advantage. However, it allows us to move to a large sample space where all the elements are trapdoored.

The compiled games are indistinguishable. We finally argue why the non-negligible advantage introduced in the first and the last hybrid does not constitute a problem: by contradiction, suppose that there exists an adversary $\mathcal{A}$ that distinguishes between the initial and the final stage with non-negligible advantage ϵ. By choosing sufficiently large parameters for the lossy mode of the ELF (which is used only in the intermediate hybrids, but not in the real and the ideal world), we can ensure that the advantage of $\mathcal{A}$ in the first and the last steps of the proof are both bounded by $\epsilon/4$. The total advantage of $\mathcal{A}$ against the compiled games would therefore be strictly smaller than ϵ, reaching a contradiction.

On the reusability of the CRS of our distributed samplers. It is easy to see that the CRS of a hardness-preserving distributed sampler is always reusable across multiple concurrent executions of the protocol. Indeed, the hardness of the search problem is not affected by the concurrent executions as the latter are always simulatable. On the other hand, the security of an indistinguishability-preserving distributed sampler can be affected by the concurrent executions. The construction presented in this paper, however, does not suffer from this issue.

2.6 Building almost everywhere extractable NIZKs

We obtain almost everywhere extractable NIZKs in the CRS model using perfectly sound NIWIs, subexponentially secure injective one-way functions, perfectly binding commitments and perfectly correct identity-based encryption (IBE).

Why consider distributed samplers that need a CRS? It may seem strange to have a distributed sampler — whose purpose is to generate a CRS — in turn rely on a CRS. What is the advantage of generating a CRS using a distributed sampler if the latter still needs a CRS? There are several reasons why a distributed sampler using a CRS can be useful: the CRS of the distributed sampler might be reused multiple times, allowing the production of many samples. The CRS of the distributed sampler protocol can also be simple to generate, perhaps because it is short or because it is unstructured (i.e. a uniform string of bits).

Our Construction. The CRS consists of an IBE master public key and a one-way function challenge v. The proofs are associated to the identity of the party that issues them. Each of them consists of a commitment c_0, an IBE encryption of the witness c_1 under the party's identity and a NIWI guaranteeing that either

c_1 contains the witness or c_0 contains the preimage of v. In order to extract the witness, it is sufficient to decrypt c_1.

Observe that, in all valid proofs for which extraction fails, the prefix is a commitment to the preimage of v. Since the one-way function is injective, the number of such prefixes depends only on the size of the randomness of the commitment scheme. As the one-way function is subexponentially secure, we can make v hard to invert even for $\mathsf{poly}(\lambda, |S|)$-time adversaries that have enough power to brute-force the commitment to retrieve the hidden value. This ensures the property we need.

Why to use identity-based encryption? In many applications of almost everywhere extractable NIZKs, we would like to argue that the programs C_0 and C_1 are indistinguishable even if we simulate the NIZKs of the honest parties (clearly, in these situations, C_1 will try to extract the witnesses only from the NIZKs of the corrupted players). The issue is that the NIZK described in the previous paragraph is not simulation-almost everywhere extractable, i.e. leaking simulated proofs may allow distinguishing between C_0 and C_1. On the other hand, disclosing C_1 might compromise the zero-knowledge property of the NIZKs due to the extraction trapdoor hidden into it.

Identity-based encryption allows us to work around the problem: to extract the witness from a NIZK proven under the identity id, we do not need the IBE master secret key, but just the private key associated to id. In other words, if we equip C_1 only with the decryption keys associated to the identities of the corrupted players, we are still able to simulate the proofs of the honest parties. The identities associated with the NIZKs guarantee that no corrupted party can publish one of the simulated proofs as it was its own.

Note that some IBE schemes such as [BF01] have uniformly random public keys. If we also use a one-way *permutation* to generate v, then the CRS is actually uniformly random. As such, our resulting distributed samplers will take a uniformly random CRS, and can be used to generate any arbitrarily structured CRS.

Theorem 6 (Informal). *Assuming perfectly correct IBE, perfectly binding non-interactive commitments, perfectly sound NIWIs and subexponential OWFs, the NIZK sketched above is almost everywhere extractable.*

2.7 CRS-less NIZKs in the Uniform Setting

All the distributed samplers we described so far make use of a CRS. The latter, needed by the NIZKs in the construction, is short, reusable and unstructured, however, is it possible to completely remove it? For indistinguishability-preserving distributed samplers, this is too much to hope for: if that was not the case, we would obtain a 3-round OT protocol with active security by compiling any 2-round OT protocol with CRS such as [PVW08]. It is known that active OT requires at least 4 rounds [HV16]. We show, however, that, if we restrict to security against uniform adversaries, we can remove the CRS from all our primitives. We obtain this by constructing CRS-less NIZKs that can be plugged in our distributed samplers.

NIZKs against uniform adversaries. The fact that NIZKs do not need CRSs if we restrict to security against uniform adversaries has been known for almost two decades: the fact was proven by Barak and Pass in [BP04] by building a CRS-less NIZK in the stand-alone model. In [BL18b], Bitansky and Lin studied a related question. They designed CRS-less NIZKs with a weak security guarantee against non-uniform adversaries: the number of false statements that can be proven is proportional to the non-uniformity of the adversary. Although this notion does not imply full soundness against uniform adversaries, it is easy to see that their constructions achieve the result. In this way, they indirectly obtain a CRS-less NIZK satisfying a weak form of simulation-soundness: a uniform adversary cannot generate proofs for false statements even if it has oracle access to the NIZK simulator that can be queried only with true statements (in the standard definition of simulation soundness, the simulator can be queried even with false statements).

Beyond these works, the topic remains rather unexplored. In this paper, we show how to construct CRS-less NIZKs achieving full simulation-soundness, simulation extractability and almost-everywhere extractability against uniform adversaries. All our constructions rely on the same trick: in order to simulate a proof, we need to use a trapdoor. Such trapdoor will be infeasible to compute for every uniform adversary but not for the simulator as it will be non-uniform.

Uniform-DDH and uniform-LWE. We start by introducing natural variations of the DDH and LWE assumptions that we believe to hold against uniform adversaries.

Consider a uniform deterministic algorithm DDHGen that outputs the description of a cyclic group $\mathbb{G}$ along with two elements $g, h \in \mathbb{G}$ such that no uniform adversary can find the value α such that $h = g^{\alpha}$. A heuristic instantiation of this algorithm is to use a SHA hash function, or the digits of π, to generate g and h. The uniform-DDH assumption states that no uniform adversary can distinguish between pairs (g^r, h^r) and pairs (g^r, h^s) where r and s are uniformly random elements. Clearly, the assumption cannot hold against non-uniform adversaries: a non-uniform adversary can receive α as part of its non-uniform advice, at that point, distinguishing is trivial. Even uniform quantum adversaries can trivially distinguish by recovering α using Shor's algorithm. We however believe that it is possible to instantiate the assumption so that all uniform, classical PPT adversaries have subexponentially small advantage.

The uniform-LWE assumption follows a similar blueprint: we use a uniform deterministic algorithm LWEGen to generate the matrix $A \in \mathbb{Z}_q^{m \times n}$ describing a lattice. We then assume that no uniform PPT adversary can distinguish $A^{\intercal} \cdot s + x$ (where s is uniform in $\mathbb{Z}_q^n$ and x is a short vector in $\mathbb{Z}_q^m$) from a random element in $\mathbb{Z}_q^m$. Once again, we cannot hope to achieve security against non-uniform adversaries: if they receive a small vector u such that $A \cdot u = 0$ as part of their non-uniform advice, they can easily break the assumption. We however believe that every uniform, classical or quantum PPT adversary has a subexponentially small advantage.

The first simulation-sound NIZKs. We obtain simulation-sound NIZKs without CRS using two different approaches. We now describe the first one.

Challengeless one-way functions. The first NIZK makes use of challengeless one-way functions (COWFs): a one-way function in which the challenge is deterministically generated by a uniform algorithm. The guarantee is that no uniform PPT adversary can find a preimage of the challenge.

We actually need two COWFs that are *independently hard*: finding preimages for any of them remains hard even when we are given a preimage for the other one. Uniform-DDH and uniform-LWE easily give a pair of independently hard one-way functions: thanks to the subexponential security of the primitive, we can make sure that, for classical adversaries, breaking uniform-DDH is strictly harder than uniform-LWE (this is achieved by making an appropriate choice of the parameters of the assumptions). On the other hand, in a post-quantum world, uniform-DDH is broken, while uniform-LWE retains its security. If breaking any of the challengeless one-way functions allows an adversary to break the other one, one of these two facts would be contradicted.

The first approach. The construction follows the blueprint of [BP04]. The proof consists of two commitments c_0 and c_1 along with a signature and a CRS-less NIWI [BOV03,GOS06a,GOS06b]. The NIZKs prove that either the statement lies in the language or one of the commitments hides a preimage for one of the independently hard challengeless one-way functions COWF_0 and COWF_1. These preimages will be used as trapdoors.

In order to achieve simulation-soundness, we need to ensure that the proof is non-malleable. We therefore generate c_0 and c_1 using a non-interactive CCA commitment without CRS [KS17,LPS17,BL18b,KK19,GKLW21]: each commitment is associated with a tag. The primitive guarantees that, given a commitment, no adversary can derive a commitment to a correlated value under a different tag. In our NIZK, similarly to [GO07], the tag will be a one-time signature verification key. Such key will be used to sign the proof. This ensures that, in order to produce a NIZK for a false statement, the adversary cannot reuse the commitments in the simulated proofs: it needs to at least change the tag (otherwise, it would need to forge a signature). The CCA security of the commitments guarantees the hardness of this task. Therefore, if the adversary manages to prove a false statement is because it discovered one of the trapdoors.

Why do we need two challengeless one-way functions? The reason is that we need to argue that the NIWIs in the simulated proofs leak no information about the trapdoors. When the statement for a simulated proof lies in the language, it is guaranteed that the NIWI does not leak the trapdoor. If that was not the case, by witness indistinguishability, the trapdoor would have been leaked even if the NIWI was generated using the witness for our statement. This contradicts the fact that the trapdoor is hard to compute. What instead if the statement does not lie in the language? In this case, the NIWI does not allow us to tell which trapdoor was used for its generation, however, it might leak some generic information about them, e.g. the minimal trapdoor according to the lexicographical order.

Using two independently hard, challengeless one-way function, we avoid this problem: by the independent hardness, if we use the COWF_0 trapdoors for the simulated proofs, the NIWIs cannot leak any COWF_1 trapdoor and vice-versa. By witness indistinguishability, we conclude that the NIWIs do not leak any of the trapdoors.

Theorem 7 (Informal). *Assuming subexponential independently secure COWFs, non-interactive CCA-commitments without CRS, subexponential CRS-less NIWIs and strong one-time signatures, the CRS-less NIZK sketched above is simulation-sound against uniform adversaries.*

The second simulation-sound NIZK. We describe the second approach to build simulation-sound NIZKs without CRS.

Labelled, challengeless one-way functions (LOWF). Our second simulation-sound NIZK makes instead use of *labelled, challengeless one-way functions* CLOWF: on input any label id, a uniform algorithm deterministically generates a one-way function challenge. The primitive guarantees that no uniform PPT adversary can invert any challenge even given the preimages associated with some of the other labels. A heuristic instantiation of this primitive can use a SHA hash function to generate the verification key for a deterministic signature scheme. In this case, the preimage associated with a label id consists of a signature on id.

The second approach. Building simulation-sound NIZKs with the second approach is perhaps even easier: each proof consists of a commitment c, a CRS-less NIWI, a signature and the relative verification key vk. The NIWI is used to prove that either the statement belongs to the language or c hides a preimage for CLOWF where the label is vk. Such preimage acts as a trapdoor.

We use a signature over the whole proof to ensure that, if the adversary manages to prove a false statement, it uses a fresh verification key (otherwise, it would have succeeded in forging a signature). That means that the adversary needs to find a preimage relative to a fresh label of CLOWF. The trapdoors used in the simulated proof do not help in this task. We can therefore achieve simulation-soundness even with malleable commitments.

Theorem 8 (Informal). *Assuming subexponential LOWF, perfectly binding non-interactive commitments, CRS-less NIWIs and strong one-time signatures, the CRS-less NIZK sketched above is simulation-sound against uniform adversaries.*

CRS-less simulation-extractable NIZK. In order to build simulation-extractable NIZKs, we introduce CRS-less non-interactive extractable commitment schemes. Observe that the primitive can exist only if we restrict to security against uniform adversaries. We build two schemes. The first one is based on uniform-DDH, the second one on uniform-LWE. A commitment consists of an encryption of the value using the public keys deterministically produced by either DDHGen or LWEGen. In the first case, we use ElGamal, in the second case, we use dual-LWE. To extract the value, it is sufficient to perform a decryption (the extractor will be a non-uniform algorithm). The operation is however infeasible for the adversary as the secret key is hard to compute in uniform polynomial-time.

In order to obtain a simulation-extractable NIZK, we simply generate an extractable commitment c to the witness for the statement we want to prove. We then use a simulation-sound NIZK to prove that c is indeed what we claim it to be.

Theorem 9 (Informal). *Assuming CRS-less simulation-sound NIZKs and subexponential CRS-less non-interactive extractable commitments, the CRS-less NIZK sketched above is simulation-extractable against uniform adversaries.*

CRS-less almost everywhere extractable NIZK. We finally present a CRS-less almost everywhere extractable NIZK with security against uniform adversaries. Differently from the construction described in Section 2.2, this NIZK will use a single extraction trapdoor for every prover's identity. On the other hand, the scheme will remain almost everywhere extractable even if we provide oracle access to the zero-knowledge simulator (we call the property *simulation-almost everywhere extractability*). This ensures that the obfuscated programs P_0 and P_1 remain indistinguishable even if the proofs of the honest parties are simulated (we recall that P_0 is a program that verifies the NIZKs proving the well-formedness of its inputs, while P_1 instead tries to extract the witnesses from them).

Independently secure labelled one-way functions and extractable commitments. The construction makes use of a labelled challengeless one-way function CLOWF and a non-interactive extractable commitment. The two primitives need to be independently secure: they need to retain their security properties even when we leak the other primitive's trapdoor. We can for instance ensure this using the same trick we adopted for simulation-sound NIZKs: we use a post-quantum extractable commitment (which can be obtained from uniform-LWE) and a quantumly-broken labelled, challengeless one-way function (heuristically, we can obtain it from any DLOG-based deterministic signature).

The reason why we need independently secure primitives is that almost everywhere extractability always requires that the simulation trapdoor (i.e. the trapdoor for CLOWF) is hard to compute in uniform polynomial time even if we leak the extraction trapdoor (i.e. the trapdoor for the extractable commitment).

On the other hand, in our construction, the proof of zero-knowledge would require the symmetric relation. Independent security allows us to satisfy both conditions simultaneously.

The simulation-almost everywhere extractable NIZK without CRS. A proof consists of two commitments c_0 and c_1, where c_1 is extractable, along with a CRS-less NIWI. The latter proves that either c_1 hides a witness for the statement we want to prove or c_0 hides a preimage for CLOWF where the label is the identity of the prover. In all the proofs where extraction fails, c_0 will therefore satisfy this second condition.

We select CLOWF so that the preimage for any given label is unique. In this way, the number of prefixes of problematic NIZKs for a given prover identity depends only on the size of the randomness of the commitment scheme. Since CLOWF is subexponentially secure, we can ensure that finding the right CLOWF preimage is infeasible even for $\mathsf{poly}\big(\lambda, |S|\big)$-time adversaries ($S$ denotes the set of problematic prefixes) that have enough computational power to recover the value hidden in c_0. Finding elements in S is therefore hard even for $\mathsf{poly}\big(\lambda, |S|\big)$-time algorithms. Learning simulated proofs under other provers' indentities does not help the adversary in the task.

Theorem 10 (Informal). *Assume the existence of a subexponential injective LOWF and a CRS-less non-interactive extractable commitment that are independently secure. Assume perfectly binding non-interactive commitments and CRS-less NIWIs. Then, the CRS-less NIZK sketched above is simulation-almost everywhere extractable against uniform adversaries.*

3 Notation and Preliminaries

In this section, we formalise the notation and recall security definitions and basic results used in this work.

Basic notation. We denote the security parameter by λ. For any $n \in \mathbb{N}$, we use $[n]$ to denote the set $\{1, 2, \ldots, n\}$. For any binary string x and integer ℓ, $\mathsf{Trunc}_{\ell}(x)$ denotes the prefix of x consisting of its first ℓ bits. Moreover, for any integers $\ell_0 < \ell_1$, we use $\mathsf{Trunc}_{\ell_0}^{\ell_1}(x)$ to denote the substring of x consisting of the bits from the ℓ_0-th position to the ℓ_1-th one. Given any NP relation $\mathcal{R}$, we denote the corresponding language by $L_{\mathcal{R}}$.

Algorithm execution. For any deterministic algorithm $\mathcal{A}$ and input x, we use the expression $a \leftarrow \mathcal{A}(x)$ to assign the output of the algorithm $\mathcal{A}$ on input x to the variable a. When $\mathcal{A}$ is probabilistic, we instead use $a \overset{\$}{\leftarrow} \mathcal{A}(x)$. Finally, if $\mathcal{A}$ is randomised, we use $a \leftarrow \mathcal{A}(x;\, r)$ to mean that a is assigned the output of $\mathcal{A}$ on input x and randomness r. If x is variable, we use $a \leftarrow x$ to assign the value of x to a. If X is a set, instead, we use $a \overset{\$}{\leftarrow} X$ to mean that a is assigned a value sampled from X uniformly at random. If $\mathcal{A}$ and $\mathcal{O}$ are algorithms, for any x and y, we use $\mathcal{A}^{\mathcal{O}(y,\cdot)}(x)$ to denote the value output by $\mathcal{A}$ on input x while having unbounded oracle access to $\mathcal{O}(y,\cdot)$. In other words, at any point in time, $\mathcal{A}$ can send values z to an oracle, which replies with $\mathcal{O}(y,z)$. We use the term *efficient distribution* to denote a uniform PPT algorithm taking only the security parameter as input.

Asymptotic behaviour. We use $\mathsf{negl}(\lambda)$ (resp. $\mathsf{nonegl}(\lambda)$) to denote a generic negligible (resp. non-negligible) function in the security parameter. Similarly, we use $\mathsf{poly}(X_1,\dots,X_n)$ to denote a generic function that is upper-bounded by a polynomial in the given variables $X_1,\dots,X_N$. Given two functions $S_0(\lambda)$ and $S_1(\lambda)$, we say that $S_0(\lambda) \ll S_1(\lambda)$ if $S_0(\lambda)$ is a $\mathsf{poly}\big(\lambda, S_1(\lambda)\big)$ function but $S_1(\lambda)$ is not $\mathsf{poly}\big(\lambda, S_0(\lambda)\big)$.

Uniform vs non-uniform adversaries. We recall that a non-uniform algorithm consists of a randomised Turing machine that, at the beginning of its execution, receives a polynomial-size advice string, whose value depends only on the security parameter. A uniform algorithm is instead a randomised Turing machine that receives no such advice string. Throughout the paper, we use AClass to denote either the class of uniform algorithms or the class of non-uniform algorithms. Observe that the latter is strictly larger than the former.

Multiparty computation. In the paper, we deal with multiparty protocols. We always assume the existence of authenticated point-to-point channels along with an authenticated broadcast medium. We often denote the i-th party by P_i. We also assume that each party is associated with a unique identity id known to all the other players. We work with static corruption and we denote the set of honest parties by H. We say that a CRS is unstructured if it is computationally indistinguishable from a uniformly random string of a given length.

Subexponential security. We say that a primitive is subexponentially secure if there exists $e > 0$ such that the advantage of every adversary running in $\mathsf{poly}\big(2^{\lambda^e}\big)$ time in the relative security game is asymptotically smaller than $2^{-\lambda^e}$.

3.1 One-Way Functions

We recall the definition of one-way function (OWF): a function that can be efficiently computed but hard to invert on random instances.

Definition 1 (One-way function). *A one-way function is a pair of uniform PPT algorithms* $(\mathsf{Gen}, \mathsf{OWF})$ *with the following syntax:*

- Gen *is randomised, takes as input the security parameter* $\mathbb{1}^\lambda$ *and outputs a pair* (v,u)*.*
- OWF *is deterministic and takes as input the security parameter* $\mathbb{1}^\lambda$ *and a value* u*. The output is a value* v*.*

We require the following properties

- ***(Correctness).*** *For every $\lambda \in \mathbb{N}$, we have*

$$\Pr\left[\mathsf{OWF}(\mathbb{1}^\lambda, u) = v \middle| (v,u) \xleftarrow{\$} \mathsf{Gen}(\mathbb{1}^\lambda)\right] = 1.$$

- ***(Security).*** *For every PPT adversary $\mathcal{A}$, we have*

$$\Pr\left[\mathsf{OWF}(\mathbb{1}^\lambda, u') = v \middle| (v,u) \xleftarrow{\$} \mathsf{Gen}(\mathbb{1}^\lambda), u' \xleftarrow{\$} \mathcal{A}(\mathbb{1}^\lambda, v)\right] = \mathsf{negl}(\lambda).$$

We say that the the one-way function is injective if

$$\Pr\left[\exists u' \neq u \quad \mathsf{OWF}(\mathbb{1}^\lambda, u') = v \middle| (v,u) \xleftarrow{\$} \mathsf{Gen}(\mathbb{1}^\lambda)\right] = 0.$$

One-way functions, including subexponentially secure ones, can be built using well studied assumptions.

3.2 Puncturable PRFs

We recall now the definition of puncturable PRF [KPTZ13,BW13,BGI14]. As for a standard PRF, it consists of a keyed functions whose outputs are indistinguishable from random as long as the key remains secret. The primitive, however, satisfies an additional property: it is possible to generate punctured keys. The latter permit evaluating the PRF in any point of its domain except for the punctured position. Furthermore, even if the punctured key is disclosed, the value of the PRF at the punctured position remains indistinguishable from random.

Definition 2 (Puncturable PRF). *Let $p(\lambda)$ and $q(\lambda)$ be polynomial functions. A puncturable PRF with input size $p(\lambda)$ and output size $q(\lambda)$ is a pair of uniform PPT algorithms $(\mathsf{Gen}, F, \mathsf{Punct})$ with the following syntax:*

- Gen *is randomised, takes as input the security parameter $\mathbb{1}^\lambda$ and outputs a key K.*
- *F is deterministic and takes as input a key K and a value $x \in \{0,1\}^{p(\lambda)}$. The output is a pseudorandom string $y \in \{0,1\}^{q(\lambda)}$.*
- Punct *is deterministic and takes as input a key K and a value $x \in \{0,1\}^{p(\lambda)}$. The output is a punctured key K^*.*

We require the following properties.

- ***(Correctness).*** *For every pair of distinct values x and x' in $\{0,1\}^{p(\lambda)}$, we have*

$$\Pr\left[F(K,x') = F(K^*,x') \middle| K \xleftarrow{\$} \mathsf{Gen}(\mathbb{1}^\lambda),\ K^* \leftarrow \mathsf{Punct}(K,x)\right] = 1.$$

- ***(Security).*** *For every pair of PPT adversaries $(\mathcal{A}_1, \mathcal{A}_2)$, we have*

$$\left| \Pr\left[\mathcal{A}_2(\psi, K^*, y_b) = b \middle| \begin{array}{l} b \xleftarrow{\$} \{0,1\} \\ K \xleftarrow{\$} \mathsf{Gen}(\mathbb{1}^\lambda) \\ (x,\psi) \xleftarrow{\$} \mathcal{A}_1(\mathbb{1}^\lambda) \\ K^* \leftarrow \mathsf{Punct}(K,x) \\ y_0 \leftarrow F(K,x) \\ y_1 \xleftarrow{\$} \{0,1\}^{q(\lambda)} \end{array} \right] - \frac{1}{2} \right| = \mathsf{negl}(\lambda).$$

Puncturable PRFs, even with subexponential security, can be easily constructed using one-way functions.

3.3 Hash Functions

We recall now the definition of collision resistant hash function. Essentially, the latter consists of a keyed function for which it is hard to find pairs of different elements that are mapped to the same value. Security relies on the unpredictability of the key. It is possible to build subexponentially secure collision resistant hash functions from well studied assumptions.

Definition 3 (Collision resistant hash function). *Let $p(\lambda)$ and $t(\lambda)$ be polynomial functions. A hash function with input size $p(\lambda)$ and digest size $t(\lambda)$ is a pair of uniform PPT algorithms* $(\mathsf{Gen}, \mathsf{Hash})$ *with the following syntax:*

- Gen *is randomised, takes as input the security parameter* $\mathbb{1}^\lambda$ *and outputs an hash key* hk*.*
- Hash *is deterministic and takes as input a hash key* hk *and a value* $x \in \{0,1\}^{p(\lambda)}$*. The output is a digest* $y \in \{0,1\}^{t(\lambda)}$*.*

We say that the hash function is collision resistant if, for PPT adversary $\mathcal{A}$, we have

$$\left| \Pr \left[\begin{matrix} x_0 \neq x_1 \\ \mathsf{Hash}(\mathsf{hk}, x_0) = \mathsf{Hash}(\mathsf{hk}, x_1) \end{matrix} \middle| \begin{matrix} \mathsf{hk} \xleftarrow{\$} \mathsf{Gen}(\mathbb{1}^\lambda) \\ (x_0, x_1) \xleftarrow{\$} \mathcal{A}(\mathbb{1}^\lambda, \mathsf{hk}) \end{matrix} \right] \right| = \mathsf{negl}(\lambda).$$

Applied cryptography makes often use of a keyless version of the above primitive for which finding collisions is still believed to be hard. We formalise the definition below. We highlight that this primitive can hope to achieve security only against uniform adversaries. Indeed, since there is no randomness involved in the construction, a non-uniform adversary can be given a collision as part of its advice string.

Definition 4 (Keyless collision resistant hash function). *Let $p(\lambda)$ and $t(\lambda)$ be polynomial functions. A keyless hash function with input size $p(\lambda)$ and digest size $t(\lambda)$ is a uniform deterministic polynomial time algorithm* KHash *that takes as input the security parameter* $\mathbb{1}^\lambda$ *and a value* $x \in \{0,1\}^{p(\lambda)}$*. The output is a digest* $y \in \{0,1\}^{t(\lambda)}$*.*

We say that the keyless hash function is collision resistant if, for every uniform *PPT adversary $\mathcal{A}$, we have*

$$\left| \Pr \left[x_0 \neq x_1, \mathsf{KHash}(\mathbb{1}^\lambda, x_0) = \mathsf{KHash}(\mathbb{1}^\lambda, x_1) \middle| (x_0, x_1) \xleftarrow{\$} \mathcal{A}(\mathbb{1}^\lambda) \right] \right| = \mathsf{negl}(\lambda).$$

3.4 Commitments

In this subsection, we recall definitions of non-interactive commitments. A non-interactive commitment scheme is a primitive that allows encoding a message

m in a string c, called the commitment. By itself, c hides the value of m, so it can be distributed to other parties without fear of revealing its secret. At a later point in time, the commitment can however be opened, disclosing the value hidden into it. The scheme guarantees the hardness of opening c to any value other than m. In other words, after the commitment is opened, the parties can be sure that who generated c had been already committed to revealing m since the time c was sent.

In this paper, we will make use of perfectly binding, computationally hiding non-interactive schemes. In particular, that means that the value hidden in the commitment remains secret only to computationally bounded adversaries. Furthermore, the commitment c uniquely determines the value hidden into it. Such schemes can be built, even with subexponential security, based on well-studied assumptions.

Definition 5 (Non-interactive commitment scheme). *Let $p(\lambda)$ be a polynomial function. A non-interactive commitment scheme with message size $p(\lambda)$ is a uniform PPT algorithm* Com *that takes as input the security parameter $\mathbb{1}^\lambda$ and a message $m \in \{0,1\}^{p(\lambda)}$. The output is a commitment c.*

We say that the scheme is perfectly binding if, for every $\lambda \in \mathbb{N}$, there exist no pairs (m_0, r_0) and (m_1, r_1) such that $m_0 \neq m_1$ and $\mathsf{Com}(\mathbb{1}^\lambda, m_0; r_0) = \mathsf{Com}(\mathbb{1}^\lambda, m_1; r_1)$.

We say that the scheme is computationally hiding if, for every pair of PPT adversaries $(\mathcal{A}_1, \mathcal{A}_2)$, we have

$$\left| \Pr\left[\mathcal{A}_2(\psi, c) = b \;\middle|\; \begin{array}{l} b \xleftarrow{\$} \{0,1\} \\ (m_0, m_1, \psi) \xleftarrow{\$} \mathcal{A}_1(\mathbb{1}^\lambda) \\ c \xleftarrow{\$} \mathsf{Com}(\mathbb{1}^\lambda, m_b) \end{array} \right] - \frac{1}{2} \right| = \mathsf{negl}(\lambda).$$

We also recall the definition of computation-enabled CCA commitment [KS17] [LPS17,BL18b,KK19,GKLW21]. This is a particular type of commitment that satisfies non-malleability. That means that given a commitment c hiding a value m, we are not able to derive another commitment c' that hides some value m' correlated to m. This property is formulated by augmenting the commitment algorithm with tags. Formally, we require that, if a value m is committed along with a tag id, m remains hidden even if the adversary has access to an inefficient oracle that extracts the values from the queried commitments. Clearly, the oracle accepts only commitments that use tags different from id.

Obtaining non-interactive non-malleable commitments with large tag space without relying on setups is not an easy task. For this reason, in this paper, we rely on constructions of this kind that achieve security only against uniform adversaries. In particular, the primitive we are interested in satisfies computation-enabled CCA security, meaning that, at the beginning of the game we described above, the uniform adversary is allowed to query a possibly inefficient, randomised Turing machine with no input. The challenger provides the adversary with the result of the machine execution.

Definition 6 (Computation-enabled CCA commitment). *Let $p(\lambda)$ and $q(\lambda)$ be polynomial functions, let $e > 0$. A e-computation enabled CCA commitment scheme with message size $p(\lambda)$ and tag size $q(\lambda)$ is a pair of uniform algorithms $(\mathsf{CCACom}, \mathsf{Val})$ with the following syntax:*

- CCACom *is PPT and takes as input the security parameter $\mathbb{1}^\lambda$, a tag $\mathsf{id} \in \{0,1\}^{q(\lambda)}$ and a message $m \in \{0,1\}^{p(\lambda)}$. The output is a commitment c.*
- Val *is deterministic and inefficient. It takes as input a label id and a commitment c and outputs either a message $m \in \{0,1\}^{p(\lambda)}$ or $\perp$.*

We require the following properties.

- ***(Correctness).*** *For every $\lambda \in \mathbb{N}$, $\mathsf{id} \in \{0,1\}^{q(\lambda)}$ and $m \in \{0,1\}^{p(\lambda)}$, we have*
$$\Pr\left[\mathsf{Val}(\mathsf{id}, c) = m \middle| c \xleftarrow{\$} \mathsf{CCACom}(\mathbb{1}^\lambda, \mathsf{id}, m)\right] = 1.$$
- ***(CCA-Hiding).*** *For every polynomials $t(\lambda)$ and $s(\lambda)$, no uniform PPT adversary $\mathcal{A}$ can win the game in Fig. 9 with non-negligible advantage.*

CCA-Hiding Game

Initialisation: This procedure is run only once, at the beginning of the game.

1. $b \xleftarrow{\$} \{0,1\}$
2. Activate $\mathcal{A}$ with $\mathbb{1}^\lambda$.
3. Receive a Turing machine P from the adversary.
4. Run P on no input for at most $t(2^{\lambda^e})$ steps. If P does not terminate before that, provide $\mathcal{A}$ with $\perp$, otherwise, provide it with the first $s(\lambda)$ bits of the output.
5. Receive a tag $\widehat{\mathsf{id}}$ from the adversary.

Value: This procedure can be queried multiple times, both before and after choosing the challenge. Upon receiving pairs (id, c) where $\mathsf{id} \neq \widehat{\mathsf{id}}$, the challenger replies with $\mathsf{Val}(\mathsf{id}, c)$.

Challenge: This procedure can be queried only once. The adversary provides $m_0, m_1 \in \{0,1\}^{p(\lambda)}$. The challenger replies with $c \xleftarrow{\$} \mathsf{CCACom}(\mathbb{1}^\lambda, \widehat{\mathsf{id}}, m_b)$.

Win: The adversary wins if it guesses b.

Fig. 9. CCA-hiding game

3.5 Strong One-Time Signatures

We recall here the definition of strong one-time signature. Informally, this consists in a signing scheme for which it is hard to craft forgeries if we are given access to just one signature. The scheme is strong in the sense that, given a signature s for a message m, it is even hard to find another signature s' for m. Strong one-time signatures can be built from one-way functions.

Definition 7 (Strong one-time signature). *Let $p(\lambda)$ be a polynomial function. A strong one-time signature is a triple of uniform PPT algorithms* $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ *with the following syntax:*

- Gen *is randomised and takes as input the security parameter $\mathbb{1}^\lambda$. The output is a key pair* $(\mathsf{vk}, \mathsf{sk})$.
- Sign *is randomised and takes as input a secret key* sk *and a message $m \in \{0,1\}^{p(\lambda)}$. The output is a signature s.*
- Verify *is deterministic and takes as input a verification key* vk*, a message $m \in \{0,1\}^{p(\lambda)}$ and a signature s. The output is a bit $b \in \{0,1\}$.*

We require the following properties.

- ***(Correctness).*** *For every $m \in \{0,1\}^{p(\lambda)}$, we have*

$$\Pr\left[\mathsf{Verify}(\mathsf{vk}, m, s) = 1 \middle| (\mathsf{vk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{Gen}(\mathbb{1}^\lambda), s \xleftarrow{\$} \mathsf{Sign}(\mathsf{sk}, m)\right] = 1.$$

- ***(Security).*** *For every pair of PPT adversaries $(\mathcal{A}_1, \mathcal{A}_2)$, we have*

$$\Pr\left[\begin{array}{c|l} & (\mathsf{vk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{Gen}(\mathbb{1}^\lambda) \\ (s, m) \neq (\widehat{s}, \widehat{m}) & (\widehat{m}, \psi) \xleftarrow{\$} \mathcal{A}_1(\mathbb{1}^\lambda, \mathsf{vk}) \\ \mathsf{Verify}(\mathsf{vk}, m, s) = 1 & \widehat{s} \xleftarrow{\$} \mathsf{Sign}(\mathsf{sk}, \widehat{m}) \\ & (s, m) \xleftarrow{\$} \mathcal{A}_2(\psi, \widehat{s}) \end{array}\right] = \mathsf{negl}(\lambda).$$

3.6 Non-Interactive Witness Indistinguishability

We recall the definition of non-interactive witness-indistinguishable proof (NIWI). Essentially, this consists of a construction specifying how to prove that a given statement x belong to a language using a single message. In order to be efficient, the algorithm that generates the proof needs to receive a witness for x as input. The primitive does not guarantee that the proof keeps the witness secret. It achieves, however, a weaker form of security stating that if there are multiple witnesses for the same statement x, the proof does not disclosed which witness was used for its generation.

It is possible to build subexponentially secure NIWI proofs without setups from various assumptions, specifically, DLIN [GOS06b,GOS06a], derandomisation [BOV03] and indistinguishability obfuscation [BP15].

Definition 8 (NIWI proof). *Let $\mathcal{R}$ be an NP relation. A NIWI proof is a pair of uniform PPT algorithms* $(\mathsf{Prove}, \mathsf{Verify})$ *with the following syntax:*

- Prove *is randomised and takes as input the security parameter $\mathbb{1}^\lambda$, a statement x and a witness w. The output is a proof π.*

- Verify *is deterministic and takes as input a proof* π *and a statement* x*. The output is a bit* $b \in \{0,1\}$*.*

We require the following properties.

- ***(Completeness).*** *There exists a negligible function* $\mathsf{negl}(\lambda)$ *such that, for every* $(x,w) \in \mathcal{R}$*, we have*

$$\Pr\left[\mathsf{Verify}(\pi,x) = 1 \middle| \pi \xleftarrow{\$} \mathsf{Prove}(\mathbb{1}^\lambda, x, w)\right] = 1 - \mathsf{negl}(\lambda).$$

- ***(Perfect Soundness).*** *If* $x \notin L_{\mathcal{R}}$*, there exists no* π *such that* $\mathsf{Verify}(\pi,x) = 1$*.*
- ***(Witness-Indistinguishability).*** *For every pair of PPT adversaries* $(\mathcal{A}_1,\mathcal{A}_2)$*, we have*

$$\left|\Pr\left[\mathcal{A}_2(\psi,\pi) = b \middle| \begin{array}{l} b \xleftarrow{\$} \{0,1\} \\ (x, w_0, w_1, \psi) \xleftarrow{\$} \mathcal{A}_1(\mathbb{1}^\lambda) \\ \pi \xleftarrow{\$} \mathsf{Prove}(\mathbb{1}^\lambda, x, w_b) \\ \textit{If } (x,w_0) \notin \mathcal{R} \textit{ or } (x,w_1) \notin \mathcal{R} : \pi \leftarrow \bot \end{array}\right] - \frac{1}{2}\right| = \mathsf{negl}(\lambda).$$

3.7 Identity-Based Encryption

We recall the definition of identity-based encryption (IBE) [Sha84,BF01]. An IBE scheme is a public-key encryption scheme that is augmented with an access policy: each ciphertext and each secret key is associated with an identity. It is possible to decrypt only if two identities match. Holding keys associated with other identities gives no help in retrieving information about the plaintext.

Definition 9 (Identity-based encryption). *Let* $p(\lambda)$ *and* $q(\lambda)$ *be polynomial functions. An identity-based encryption scheme (IBE) with message size* $p(\lambda)$ *and identity size* $q(\lambda)$ *is a tuple of uniform PPT algorithms* $(\mathsf{Setup}, \mathsf{Extract}, \mathsf{Enc}, \mathsf{Dec})$ *with the following syntax:*

- Setup *is randomised and takes as input the security parameter* $\mathbb{1}^\lambda$*. The output is a key pair* $(\mathsf{mpk}, \mathsf{msk})$*.*
- $\mathsf{Extract}$ *is randomised and takes as input a master secret key* msk *and an identity* $\mathsf{id} \in \{0,1\}^{q(\lambda)}$*. The output is a secret-key* sk*.*
- Enc *is randomised and takes as input a master public key* mpk*, an identity* $\mathsf{id} \in \{0,1\}^{q(\lambda)}$ *and a message* $m \in \{0,1\}^{p(\lambda)}$*. The output is a ciphertext* c*.*
- Dec *is deterministic and takes as input a secret-key* sk *and a ciphertext* c*. The output is a message* m *or* $\bot$*.*

We require the following properties.

- ***(Perfect Correctness).*** *For every* $\mathsf{id} \in \{0,1\}^{q(\lambda)}$ *and* $m \in \{0,1\}^{p(\lambda)}$*,*

$$\Pr\left[\mathsf{Dec}(\mathsf{sk}, c) = m \middle| \begin{array}{l} (\mathsf{mpk}, \mathsf{msk}) \xleftarrow{\$} \mathsf{Setup}(\mathbb{1}^\lambda) \\ c \xleftarrow{\$} \mathsf{Enc}(\mathsf{mpk}, \mathsf{id}, m) \\ \mathsf{sk} \xleftarrow{\$} \mathsf{Extract}(\mathsf{msk}, \mathsf{id}) \end{array}\right] = 1.$$

- ***(IND-ID-CPA security)****. No PPT adversary can win the game in Fig. 10 with non-negligible advantage.*

The IND-ID-CPA Game

Initialisation: This procedure is run only once, at the beginning of the game.

1. $b \xleftarrow{\$} \{0,1\}$
2. $Q \leftarrow \emptyset$
3. $\widehat{\mathsf{id}} \leftarrow \perp$
4. $(\mathsf{mpk}, \mathsf{msk}) \xleftarrow{\$} \mathsf{Setup}(\mathbb{1}^\lambda)$
5. Activate the adversary with $\mathbb{1}^\lambda$ and mpk.

Key: This procedure can be queried multiple time, both before and after choosing the challenge. On input identities $\mathsf{id} \in \{0,1\}^{q(\lambda)}$ such that $\mathsf{id} \neq \widehat{\mathsf{id}}$, the challenger adds id to Q and replies with $\mathsf{Extract}(\mathsf{msk}, \mathsf{id})$.

Challenge: This procedure can be queried only once. The adversary provides $m_0, m_1 \in \{0,1\}^{p(\lambda)}$ and $\widehat{\mathsf{id}} \in \{0,1\}^{q(\lambda)} \setminus Q$. The challenger answers with $\mathsf{Enc}(\mathsf{mpk}, \widehat{\mathsf{id}}, m_b)$.

Win: The adversary wins if it guesses b.

Fig. 10. The IND-ID-CPA game

Subexponentially secure IBE schemes can be built in the plain model using a large variety of assumptions [CHK03,BB04,Wat05,Gen06,ABB10].

3.8 Indistinguishability Obfuscation

We recall the definition of indistinguishability obfuscation [BGI$^+$01,GGH$^+$13]. An indistinguishability obfuscation is an algorithm that modifies a circuit without altering its input-output behaviour. The result is however so "scrambled" that it is hard to tell what the original circuit looked like. In this paper, we use the terms "circuit" and "program" interchangeably.

Definition 10 (Indistinguishability obfuscator). *An indistinguishability obfuscator is a uniform PPT algorithm* iO *that takes as input the security parameter* $\mathbb{1}^\lambda$ *and a circuit* C*. The output is an obfuscate program* $\widetilde{C}$*. We require the following properties.*

- ***(Perfect Correctness).*** *For every circuit* C *and input* x*, we have*

$$\Pr\left[C(x) = \widetilde{C}(x) \middle| \widetilde{C} \xleftarrow{\$} \mathsf{iO}(\mathbb{1}^\lambda, C)\right] = 1.$$

- ***(Security).*** *For every PPT adversary* $\mathcal{A}$ *and sampler* Samp *outputting same-size circuits* C_0 *and* C_1 *such that* $\forall x : C_0(x) = C_1(x)$ *along with auxiliary* *information* aux*, we have*

$$\left| \Pr\left[\mathcal{A}(\mathbb{1}^\lambda, \widetilde{C}, C_0, C_1, \mathsf{aux}) = b \middle| \begin{array}{l} b \xleftarrow{\$} \{0,1\} \\ (C_0, C_1, \mathsf{aux}) \xleftarrow{\$} \mathsf{Samp}(\mathbb{1}^\lambda) \\ \widetilde{C} \xleftarrow{\$} \mathsf{iO}(\mathbb{1}^\lambda, C_b) \end{array} \right] - \frac{1}{2} \right| = \mathsf{negl}(\lambda).$$

Although the initial obfuscation constructions were based on non-standard assumptions [GGH$^+$13], the field has recently shown significant progress. State-of-the-art obfuscators can indeed be based on the subexponential hardness of well-founded problems [JLS21,JLS22]. Notice that the subexponential security of obfuscation is a common assumption in cryptography [CLTV15,DHRW16,HIJ$^+$17].

In this paper, we will use indistinguishability obfuscators satisfying a particular property called injectivity. In other words, it is guaranteed that the obfuscation of distinct circuits will never collide. It is easy to obtain this property by appending a perfectly binding commitment of the unobfuscated circuit to the obfuscated program [CCK$^+$22].

Definition 11 (Injective indistinguishability obfuscator). *We say that an indistinguishability obfuscator* iO *is injective if, for every* $\lambda \in \mathbb{N}$*, there exist no pairs* (C_0, r_0) *and* (C_1, r_1) *such that* $C_0 \neq C_1$ *but* $\mathsf{iO}(\mathbb{1}^\lambda, C_0;\, r_0) = \mathsf{iO}(\mathbb{1}^\lambda, C_1;\, r_1)$*.*

3.9 Multi-Key FHE

We recall the definition of multi-key fully homomorphic encryption [LTV12,CM15] [MW16]. As standard FHE, multi-key FHE scheme is a public key encryption scheme that allows homomorphically applying functions on encrypted values deriving encryptions of the outputs. The evaluation of the function is performed locally and no information about the plaintexts is revealed in the process. The big advantage of multi-key FHE is that, while standard FHE allows performing operations only between ciphertexts encrypted under the same public key, multi-key FHE suffers from no such restriction: we can evaluate functions on inputs encrypted under different keys, obtaining an encryption of the output under a "joint key". In order to decrypt the latter, the parties need to collaborate: each player will locally compute a partial decryption using its own private key. By pooling together the partial plaintexts, everybody can retrieve the result.

Subexponentially secure multi-key FHE without CRS can be built based on LWE and DSPR [AJJM20], or obfuscation and DDH [DHRW16]. In this paper, we rely on the definition of [AJJM20].

Definition 12 (Multi-key FHE). *An multi-key fully homomorphic encryption scheme is a tuple of uniform PPT algorithms* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{PartDec}, \mathsf{FinDec})$ *with the following syntax:*

- Gen *is randomised and takes as input the security parameter* $\mathbb{1}^\lambda$*. The output is a key pair* $(\mathsf{pk}, \mathsf{sk})$*.*

- Enc *is randomised and takes as input a public key* pk *and a message* m*. The output is a ciphertext* c*.*
- Eval *is deterministic and takes as input a function* f *and* n *pairs* (pk_i, c_i) *where* n *is the number of inputs of* f*. The output is a ciphertext* C *encrypted under the joint public key* $(\mathsf{pk}_1, \ldots, \mathsf{pk}_n)$*.*
- $\mathsf{PartDec}$ *is randomised and takes as input a ciphertext* C*,* n *public keys* $\mathsf{pk}_1, \ldots, \mathsf{pk}_n$ *for some* $n \in \mathbb{N}$*, an index* $i \in [n]$ *and a secret key* sk*. The output is a partial decryption* d*.*
- FinDec *is deterministic and takes as input* n *partial decryptions* $d_1, \ldots, d_n$ *for some* $n \in \mathbb{N}$*. The output is a message* m *or* $\perp$*.*

We require the following properties.

- ***(Correctness).*** *For every function* f *with* n *inputs and values* $x_1, \ldots, x_n$*, we have*

$$\Pr\left[m = f(x_1, \ldots, x_n) \middle| \begin{array}{l} \forall i \in [n] : (\mathsf{pk}_i, \mathsf{sk}_i) \xleftarrow{\$} \mathsf{Gen}(1\!\!1^\lambda) \\ \forall i \in [n] : c_i \xleftarrow{\$} \mathsf{Enc}(\mathsf{pk}_i, x_i) \\ C \leftarrow \mathsf{Eval}(f, \mathsf{pk}_1, c_1, \ldots, \mathsf{pk}_n, c_n) \\ \forall i \in [n] : d_i \xleftarrow{\$} \mathsf{PartDec}(C, \mathsf{pk}_1, \ldots, \mathsf{pk}_n, i, \mathsf{sk}_i) \\ m \leftarrow \mathsf{FinDec}(d_1, \ldots, d_n) \end{array} \right] = 1.$$

- ***(Reusable Semi-Malicious Security).*** *There exists uniform PPT simulators* Sim_1 *and* Sim_2 *such that no PPT adversary* $\mathcal{A}$ *can win the game in Fig. 11 with non-negligible advantage.*

3.10 Extremely Lossy Functions

We recall the definition of extremely lossy function (ELF) [Zha16]. An ELF is a function f parametrised by two values M and r. The former denotes the cardinality of its domain, whereas r denotes an upper bound on the size of the image. When $M = r$, the function is guaranteed to be injective. When $r \neq M$, we say that the ELF is in lossy mode. The primitive ensures that, by choosing a sufficiently large $\mathsf{poly}(\log M)$ value r, the advantage in distinguishing between an injective ELF and a lossy ELF can be made an arbitrarily small inverse polynomial function in $\log M$. Extremely lossy functions can be built based on the *exponential* hardness of DDH over elliptic curves [Zha16].

Definition 13 (Extremely lossy function). *An extremely lossy function (ELF) consists of a uniform PPT algorithm* Gen *that takes as input two integers* M *and* r*. The output is the description of a function* f *with domain* $[M]$*. The primitive uses* $\log M$ *as security parameter. We require the following properties.*

- f *is computable in* $\mathsf{poly}\log(M)$ *time and the running time is independent of* r*.*
- *If* $r = M$*,* $\Pr\left[\exists x \neq y \text{ s.t. } f(x) = f(y) \middle| f \xleftarrow{\$} \mathsf{Gen}(M, M)\right] = \mathsf{negl}(\log M)$*.*

The Multi-Key FHE Security Game

```
Initialisation: This procedure is run only once, at the beginning.
 1. $b \xleftarrow{\$} \{0,1\}$
 2. Activate the adversary $\mathcal{A}$ with $\mathbb{1}^\lambda$
 3. Receive $H \subseteq [n]$ from the adversary along with $(x_i)_{i \in H}$.
 4. $\forall i \in H : (\mathsf{pk}_i^0, \mathsf{sk}_i^0) \xleftarrow{\$} \mathsf{Gen}(\mathbb{1}^\lambda, i)$
 5. $\forall i \in H : c_i^0 \xleftarrow{\$} \mathsf{Enc}(\mathsf{pk}_i^0, x_i)$
 6. $\big(\phi, (\mathsf{pk}_i^1, c_i^1)_{i \in H}\big) \xleftarrow{\$} \mathsf{Sim}_1(\mathbb{1}^\lambda, H)$
 7. $\forall i \in H : (\mathsf{pk}_i, c_i) \leftarrow (\mathsf{pk}_i^b, c_i^b)$
 8. Provide $\mathcal{A}$ with $(\mathsf{pk}_i, c_i)_{i \in H}$.
 9. Receive $(x_j, r_j, r'_j)_{j \notin H}$ from $\mathcal{A}$
10. $\forall j \notin H : (\mathsf{pk}_j, \mathsf{sk}_j) \leftarrow \mathsf{Gen}(\mathbb{1}^\lambda, j;\, r_j)$
11. $\forall j \notin H : c_j \leftarrow \mathsf{Enc}(\mathsf{pk}_j, x_j;\, r'_j)$
Decryption: This procedure can be queried multiple times. On input a function
$f$ with $n$ inputs, compute the following.
 1. $C \leftarrow \mathsf{Eval}(f, \mathsf{pk}_1, c_1, \ldots, \mathsf{pk}_n, c_n)$
 2. $y \leftarrow f(x_1, \ldots, x_n)$
 3. $\forall i \in H : d_i^0 \xleftarrow{\$} \mathsf{PartDec}(C, \mathsf{pk}_1, \ldots, \mathsf{pk}_n, i, \mathsf{sk}_i^0)$
 4. $\big(\phi', (d_i^1)_{i \in H}\big) \xleftarrow{\$} \mathsf{Sim}_2\big(\phi, f, y, (x_j, r_j, r'_j)_{j \notin H}\big)$
 5. $\phi \leftarrow \phi'$
 6. Provide $(d_i^b)_{i \in H}$ to $\mathcal{A}$
Win: The adversary wins if it guesses $b$.
```

Fig. 11. The multi-key FHE security game

- *There every $r \in [M]$,* $\Pr\Big[\big|f\big([M]\big)\big| \geq r \Big| f \xleftarrow{\$} \mathsf{Gen}(M, r)\Big] = \mathsf{negl}(\log M)$.
- *For every polynomial p and inverse polynomial function δ, there exists a polynomial q such that, for every adversary $\mathcal{A}$ running in time at most $p(\log M)$, and $r \geq q(\log M)$, we have*

$$\left| \Pr\left[\mathcal{A}(M, r, f_b) = 1 \middle| \begin{array}{l} b \xleftarrow{\$} \{0,1\} \\ f_1 \xleftarrow{\$} \mathsf{Gen}(M, r) \\ f_0 \xleftarrow{\$} \mathsf{Gen}(M, M) \end{array} \right] - \frac{1}{2} \right| \leq \delta(\log M).$$

In the constructions in this paper, $\log M$ will be both $\mathsf{poly}(\lambda)$ and $\Omega(\lambda)$. Therefore, every negligible function in $\log M$ will also be negligible in λ (and viceversa). Similarly, every polynomial function in $\log M$ will also be polynomial in λ (and viceversa).

We now recall the definition of regular ELF [Zha16]. Informally, an ELF is regular if, by applying the function on a random domain element, we hit all the elements in the image with at least inverse-polynomial probability in r and $\log M$. Regular ELFs can be built based on exponential DDH [Zha16].

Definition 14 (Regular ELF). *An ELF is regular if there exists $s = \mathsf{poly}(\log M, r)$ such that, except with negligible probability over $f \xleftarrow{\$} \mathsf{Gen}(M, r)$, for every $y \in f([M])$, we have*

$$\Pr_x\big[f(x) = y \big| x \xleftarrow{\$} [M]\big] \geq \frac{1}{s(\log M, r)}$$

where $\Pr_x$ is a probability taken over the randomness of x.

We also recall the definition of strongly efficiently enumerable ELF [Zha16]. This consists an ELF in which it is possible to reconstruct the image in $\mathsf{poly}(\log M, r)$ time.

Definition 15 (Strongly efficiently enumerable ELF). *An ELF is strongly efficiently enumerable if there exists a randomise algorithm* Enum *running in* $\mathsf{poly}(\log M, r)$ *time such that, for every $r \in [M]$,*

$$\Pr\Big[S \neq f\big([M]\big) \Big| f \xleftarrow{\$} \mathsf{Gen}(M, r), S \xleftarrow{\$} \mathsf{Enum}(M, \mathbb{1}^r, f)\Big] \leq \mathsf{negl}(\log M).$$

It easy to show that every regular ELF is strongly efficiently enumerable [Zha16].

Theorem 11 ([Zha16]). *A regular ELF is strongly efficiently enumerable.*

3.11 Distributed Samplers

Distributed samplers [ASY22] are a powerful primitive allowing n parties to securely generate a sample from a fixed distribution $\mathcal{D}(\mathbb{1}^\lambda)$ using a single round of interaction. A natural application of these constructions is the distributed generation of (structured or unstructured) common reference strings in one round.

In [ASY22], the authors showed that distributed samplers can also be used to build public-key PCFs [OSY21,ASY22], a primitive producing large amounts of correlated randomness with minimal communication and a single round of interaction.

Known constructions. The notion of distributed sampler was introduced for the first time in [ASY22]. In their work, Abram, Scholl and Yakoubov showed how to build distributed samplers for any efficiently samplable distribution $\mathcal{D}(\mathbb{1}^\lambda)$ using strong cryptographic primitives such as polynomially secure indistinguishability obfuscation [BGI$^+$01,GGH$^+$13] and a weaker form of multi-key FHE called multiparty homomorphic encryption (MHE) [AJJM20,MW16]. The authors achieved constructions in the plain model with security against non-rushing semi-malicious adversaries[7], statically corrupting up to $n-1$ parties. In such setting, distributed samplers were defined as one-round protocols that implement the functionality that generates a sample from the distribution $\mathcal{D}(\mathbb{1}^\lambda)$ and outputs it to all the parties.

The authors focussed on active security too. They managed to upgrade their constructions to this setting, unfortunately, at the price of relying on random oracles. Active distributed samplers were defined as one-round protocols that implement the functionality $\mathcal{F}_\mathcal{D}$ (see Fig. 12) in the UC model. Such functionality provides the adversary with a polynomial number of samples from $\mathcal{D}(\mathbb{1}^\lambda)$ and lets it choose the one it likes the most as the final output of the protocol. Although $\mathcal{F}_\mathcal{D}$ allows influence to the adversary, the functionality is strong enough to generate CRSs for MPC protocols without compromising security.

The active distributed sampler functionality $\mathcal{F}_\mathcal{D}$

Sample. On input Sample from the adversary, compute $R \xleftarrow{\$} \mathcal{D}(\mathbb{1}^\lambda)$ and reply with R and a unique identifier id. The adversary can query this procedure multiple times.

Output. On input $\widehat{\mathsf{id}}$ from the adversary, the functionality retrieves the sample with identifier $\widehat{\mathsf{id}}$ and outputs it to all honest parties. If such sample is not defined, the functionality aborts.

Fig. 12. The functionality for active distributed samplers in [ASY22]

Known impossibilities. A recent work by Abram, Obremski and Scholl [AOS23] proved that, without random oracle, it is impossible to build non-trivial active

[7] Similarly to the semi-honest case, a semi-malicious adversary is forced to follow the protocol, but it can choose the randomness tapes of the corrupted players as it prefers. Since the adversary is non-rushing, the choice of the randomness must be taken at the beginning of the protocol, before the honest messages are delivered.

distributed samplers satisfying the definition of [ASY22]. Actually, the impossibility holds even if we try to achieve security against *rushing*, semi-malicious adversaries.

Abram, Obremski and Scholl started by showing that active distributed samplers always need common reference strings. Then, they proved that such CRSs cannot be reused more than once, they cannot be significantly shorter than the Yao entropy of the distribution $\mathsf{H}_{\mathsf{Yao}}(\mathcal{D})$ (they can be at most $O(\log \lambda)$ bits shorter) and they cannot be unstructured (unless $\mathcal{D}$ is obliviously samplable[8]). These results, which just assume the existence of OWFs, suggest that, without random oracles, active distributed samplers cannot improve upon the trivial construction in which we directly encode a sample from $\mathcal{D}(\mathbb{1}^\lambda)$ in the CRS. In this work, we present how to get around these impossibilities by weakening the security definition of distributed sampler.

4 Almost Everywhere Extractable NIZKs

The main purpose of distributed sampler is to generate secure CRSs for multiparty computation protocols using a single round of interaction. As we discussed in the introduction, distributed samplers can be interesting if they rely on CRSs as long as the latter have nice properties such as reusability, short length and unstructuredness.

The distributed sampler we present in this paper will make use of particular NIZKs that, if instantiated with constructions from previous works, would compromise the reusability of the CRS. In this section, we formalise the security properties we require from these primitives. Furthermore, we explain how to realise our definitions obtaining short and unstructured CRSs that do not compromise reusability.

Performing extractions inside obfuscated programs. We describe the context in which we would like to use our NIZKs. We start from a NIZK satisfying black-box straight-line extraction. We consider an obfuscated program C_0 that receives a NIZK proof π among its input, verifies it and, based on the result, either outputs $\perp$ (when the verification fails) or performs other operations. We would like to argue that this obfuscated circuit is indistinguishable from another obfuscated circuit C_1 that has an extraction trapdoor hardcoded. When C_1 receives π as input, it not only verifies the proof, but it also tries to extract the corresponding witness. If any of the procedures fails, C_1 outputs $\perp$, otherwise, it performs the same operations as C_0.

Since it is hard for the adversary to come up with a proof that verifies but cannot be extracted, one could hope to prove indistinguishability between C_0 and C_1 using obfuscation. Unfortunately, we cannot rely on iO as, due to zero-knowledge, C_0 and C_1 will always have differing inputs. Specifically, we know

[8] A distribution is obliviously samplable if given a sample R from $\mathcal{D}(\mathbb{1}^\lambda)$, we can simulate the randomness that produced R. In other words, we can securely generate samples but directly feeding public random coins into $\mathcal{D}(\mathbb{1}^\lambda)$.

that simulated proofs exist, verify, but cannot be extracted, so they immediately lead to differing inputs.

The only way to avoid this problem is to rely on constructions in which the CRS only allows simulating proofs for a fixed set of statements S having polynomial size $p(\lambda)$. This idea was for instance used in [HIJ$^+$17]. With this trick, we could augment the extraction trapdoor with a list of witnesses for the statements in S, so the extraction from simulated proof will never fail. This solution, however, has the disadvantage of letting the CRS grow with $p(\lambda)$. That would make the CRS of our distributed sampler long and would hinder reusability.

Differing-input obfuscation would solve our problems. We consider diO. This primitive guarantees the hardness in distinguishing between the obfuscation of two circuits as long as differing inputs are hard to find. Although the existence of general-purpose diO for circuits is often doubted [GGHW14,BSW16], we know that, for some classes of circuits, indistinguishability obfuscators are also differing-input obfuscators. In particular, in [BCP14], Boyle, Chung and Pass proved that this is the case when the number of differing inputs is polynomial. By relying on subexponential secure obfuscation, it is easy to generalise the result of [BCP14] as follows.

Lemma 1 ([BCP14]). *Let* Samp *be a probabilistic algorithm outputting two circuits* C_0*,* C_1 *with input space* $\{0,1\}^{m(\lambda)}$*, auxiliary information* aux *and a secret* ρ*. Let* $\mathcal{O}$ *be another probabilistic algorithm that on input a pair* (ρ, x) *outputs a value* y*. Suppose the following*

- *there exist efficiently computable values* $\ell_0(C_0, C_1, \mathsf{aux}), \ell_1(C_0, C_1, \mathsf{aux})$ *and* $d(\lambda)$ *(the latter potentially superpolynomial) such that*
$$\Pr\left[\left|\mathsf{DI}^{\ell_0,\ell_1}_{C_0,C_1}\right| \leq d(\lambda) \middle| (C_0, C_1, \mathsf{aux}, \rho) \xleftarrow{\$} \mathsf{Samp}(\mathbb{1}^\lambda)\right] = 1 - \mathsf{negl}(\lambda),$$
where $\mathsf{DI}^{\ell_0,\ell_1}_{C_0,C_1} := \left\{\mathsf{Trunc}^{\ell_1}_{\ell_0}(x) \middle| C_0(x) \neq C_1(x)\right\}$.
- *for every probabilistic adversary* $\mathcal{A} \in \mathsf{AClass}$ *running in* $\mathsf{poly}(\lambda, d(\lambda))$ *time,*
$$\Pr\left[y \in \mathsf{DI}^{\ell_0,\ell_1}_{C_0,C_1} \middle| \begin{array}{l} (C_0, C_1, \mathsf{aux}, \rho) \xleftarrow{\$} \mathsf{Samp}(\mathbb{1}^\lambda) \\ y \xleftarrow{\$} \mathcal{A}^{\mathcal{O}(\rho,\cdot)}(\mathbb{1}^\lambda, \mathbb{1}^{d(\lambda)}, C_0, C_1, \mathsf{aux}) \end{array}\right] = \mathsf{negl}(\lambda).$$

Let iO *be an indistinguishability obfuscator against which every PPT adversary has advantage at most* $\mathsf{negl}(\lambda)/d(\lambda)$*. Then, for every PPT adversary* $\mathcal{A} \in \mathsf{AClass}$ *we have*

$$\left|\Pr\left[\mathcal{A}^{\mathcal{O}(\rho,\cdot)}(\mathbb{1}^\lambda, C_0, C_1, \widetilde{C}, \mathsf{aux}) = b \middle| \begin{array}{l} b \xleftarrow{\$} \{0,1\} \\ (C_0, C_1, \mathsf{aux}, \rho) \xleftarrow{\$} \mathsf{Samp}(\mathbb{1}^\lambda) \\ \widetilde{C} \xleftarrow{\$} \mathsf{iO}(\mathbb{1}^\lambda, C_b) \end{array}\right] - \frac{1}{2}\right| = \mathsf{negl}(\lambda).$$

Sketch of the proof. The proof follows the blueprint of [BCP14][Theorem 6.2], in which they convert a successful PPT distinguisher $\mathcal{A}$ into a successful extractor for (prefixes of) differing inputs. The only differences is that now, $\mathcal{A}$

has unbounded access to $\mathcal{O}(\rho, \cdot)$ and we are looking for substrings of differing-inputs so the binary search will involve only the bits of the inputs in between position $\ell_0(C_0, C_1, \mathsf{aux})$ and $\ell_1(C_0, C_1, \mathsf{aux})$. Furthermore, the parameter $d(\lambda)$ is potentially superpolynomial.

Observe that the extractor of [BCP14][Lemma 6.3] is uniform if $\mathcal{A}$ is uniform. Notice indeed that the extractor does not need to know the advantage $\epsilon(\lambda)$ of $\mathcal{A}$ [9], but only the polynomial $p(\lambda)$ such that, for every $\lambda' \in \mathbb{N}$, there exists $\lambda'' \geq \lambda'$ such that $\epsilon(\lambda'') \geq 1/p(\lambda'')$. Moreover, it runs in time is at most $d^3(\lambda) \cdot \mathsf{poly}(\lambda)$. Finally, it still outputs a prefix of differing-input with non-negligible probability. □

[9] This could be impossible when the extractor is uniform

Our goal will be to build particular NIZKs that will allow us to apply the above lemma. More in detail, we want that the prefix of all proofs that verify but cannot be extracted lies in a set VPFE whose elements are hard to compute even for adversaries running in time $d(\lambda) := |\mathsf{VPFE}|$. If we succeed in doing this, assuming the subexponential hardness of iO, we can argue that the obfuscation of C_0 and C_1 are indistinguishable despite the existence of differing inputs. We call the NIZK satisfying this particular property *almost everywhere extractable NIZKs.*

Almost everywhere extractable NIZKs. In this section, we formalise the properties of the NIZK needed by our distributed samplers. We recall here the definition of identity-based NIZK [KOR05]. Informally, this is a primitive in which both the proving and the verification algorithms are augmented with an input id denoting an identity. Completeness is guaranteed only if the algorithms use the same id.

Definition 16 (Identity-based NIZK). *Let $\mathcal{R}$ be an NP relation. An identity-based NIZK for $\mathcal{R}$ is a triple of uniform PPT algorithms* $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ *with the following syntax*

- Setup *is randomised and takes as input the security parameter and outputs a CRS σ.*
- Prove *is randomised and takes as input the security parameter, the CRS σ, an identity* id*, a statement x and the corresponding witness w. The output is a proof π.*
- Verify *is deterministic and takes as input the CRS σ, an identity* id*, a proof π and a statement x. The output is a bit b representing whether the statement was accepted or not.*

We require that the construction satisfies completeness, namely that there exists a negligible function $\mathsf{negl}(\lambda)$ *such that, for every $(x, w) \in \mathcal{R}$ and identity* id*,*

$$\Pr\left[\mathsf{Verify}(\sigma, \mathsf{id}, \pi, x) = 1 \middle| \begin{array}{l} \sigma \xleftarrow{\$} \mathsf{Setup}(\mathbb{1}^\lambda) \\ \pi \xleftarrow{\$} \mathsf{Prove}(\mathbb{1}^\lambda, \sigma, \mathsf{id}, x, w) \end{array}\right] \geq 1 - \mathsf{negl}(\lambda).$$

Why do we need identity-based NIZKs? Almost everywhere extractable NIZKs will be identity-based. We recall that our goal is to design NIZKs for which it is difficult to distinguish an obfuscated program that simply verifies the provided proofs and one that instead tries to extract the witnesses. Now, in the security proofs of many applications, e.g. our distributed samplers, the adversary will be given many simulated proofs. In general, these are proofs where extraction fails although the verification succeeds! If we feed any of these proofs to the obfuscated programs, we can trivially discover if the circuit tries to extract the witness or not (in the first case, the output will always be $\perp$).

Identity-based NIZKs allow us to find a way around the problem: we modify the programs so that they will only accept proofs that verify with respect to specific hardcoded identities. If the identities of the simulated proofs differ from the hardcoded ones, the behaviour of the program on input these simulated proofs will be independent of whether extraction if performed or not. In order words, we are using identities to restrict the scope of the proofs.

Alternative approaches. There are two ways we can proceed towards our goal. The first one is to achieve a stronger form of simulation-extractability: forging a valid proof where extraction fails must be hard even if we provide simulated proofs for different identities. Although we use this approach in Section 10 to build almost everywhere extractable NIZKs with security against uniform adversaries, in this section, we adopt a different solution: we strengthen the notion of zero-knowledge. In particular, the extraction will take place in two steps: first, from the general extraction trapdoor and the identity associated with the proof, we derive an identity-specific trapdoor. Then, we use the latter to extract the witness. We require that zero-knowledge holds even if we leak identity-specific extraction trapdoors where the underlying identities differ from those of the simulated proofs. The obfuscated programs will contain only extraction trapdoors associated with their hardcoded identities.

The two approaches lead to different proving strategies. If we rely on simulation-extractability, the security proof of our application will first consider the hybrid in which the NIZKs of the honest parties are simulated and then we switch to obfuscated programs that try to extract witnesses. If we strengthen zero-knowledge, we will do the opposite: first, we switch to programs that extract the witnesses and then we simulate the proofs of the honest players. The results are equivalent.

In this section, we decided to follow the second approach as it allows us to achieve our goal under weaker assumptions. Following the blueprint in Section 10, it would also have been possible to adopt the first approach, however, that would require assuming the existence of $B(\lambda)$-bounded labelled one-way functions (that do not need to be challengeless) that are secure against adversaries running in $\mathsf{poly}\big(2^{\lambda^e}, B(\lambda)\big)$ for some $e > 0$.

Defining almost-everywhere extractability. We proceed by formalising the definition of almost everywhere extractable NIZK. The construction relies on two trapdoors τ_s and τ_e, the first one will be used to simulate the proofs, the second one will be used to extract the witnesses. The extraction is divided into two procedures: given a proof π with underlying identity id, we first derive the extraction trapdoor associated with id using $\tau_e^{\mathsf{id}} \xleftarrow{\$} \mathsf{Trap}(\tau_e, \mathsf{id})$. Next, we extract the witness from π using τ_e^{id}. It is straightforward to see that an almost everywhere extractable NIZK is also a non-interactive argument of knowledge.

Definition 17 (Almost everywhere extractable NIZK). *An identity-based NIZK for the NP relation $\mathcal{R}$ is* almost everywhere extractable *if there exists a uniform PPT algorithms* $\mathsf{SimSetup}$*,* Trap *and* $\mathsf{Extract}$ *with the following properties*

– *No PPT adversary can distinguish between*

$$\left\{\sigma \middle| \sigma \xleftarrow{\$} \mathsf{Setup}(\mathbb{1}^\lambda)\right\} \qquad \left\{\sigma \middle| (\sigma, \tau_s, \tau_e) \xleftarrow{\$} \mathsf{SimSetup}(\mathbb{1}^\lambda)\right\}$$

– *The algorithm* Extract *is deterministic and, for every* $w = \mathsf{Extract}(\tau_e^{\mathsf{id}}, \pi, x)$*,*

$$\Pr\left[(x, w) \in \mathcal{R} \middle| w \neq \bot\right] = 1.$$

– *There exist values* $\ell(\lambda) \in [m]$ *and* $d(\lambda)$ *(the latter potentially superpolynomial) and a negligible function* $\mathsf{negl}(\lambda)$ *such that, for every identity* id*,*

$$\Pr\left[\left|\mathsf{VPFE}_{\sigma,\tau_e,\mathsf{id}}\right| \leq d(\lambda) \middle| (\sigma, \tau_s, \tau_e) \xleftarrow{\$} \mathsf{NIZK.SimSetup}(\mathbb{1}^\lambda)\right] \geq 1 - \mathsf{negl}(\lambda),$$

where

$$\mathsf{VPFE}_{\sigma,\tau_e\,\mathsf{id}} := \left\{ \mathsf{Trunc}_\ell(\pi) \middle| \exists (x, r) \ s.t. \begin{array}{r} \mathsf{NIZK.Verify}(\sigma, \mathsf{id}, \pi, x) = 1 \\ \mathsf{NIZK.Trap}(\tau_e, \mathsf{id};\ r) = \tau_e^{\mathsf{id}} \\ \mathsf{NIZK.Extract}(\tau_e^{\mathsf{id}}, \pi, x) = \bot \end{array} \right\}$$

– *For every probabilistic adversary* $\mathcal{A}$ *running in* $\mathsf{poly}(\lambda, d(\lambda))$ *time, there exits a negligible function* $\mathsf{negl}(\lambda)$ *such that, for every identity* id

$$\Pr\left[y \in \mathsf{VPFE}_{\sigma,\tau_e,\mathsf{id}} \middle| \begin{array}{l} (\sigma, \tau_s, \tau_e) \xleftarrow{\$} \mathsf{NIZK.SimSetup}(\mathbb{1}^\lambda) \\ y \xleftarrow{\$} \mathcal{A}(\mathbb{1}^\lambda, \mathbb{1}^{d(\lambda)}, \sigma, \tau_e) \end{array} \right] \leq \mathsf{negl}(\lambda).$$

We prove below that, if we use almost everywhere extractable NIZKs and we rely on a subexponentially secure iO scheme, the obfuscation of the programs C_0 and C_1 are indistinguishable.

Lemma 2. *Let* NIZK *be an almost everywhere extractable NIZK for the relation* $\mathcal{R}$*. Let* $d(\lambda)$ *be the upper-bound on* $\left|\mathsf{VPFE}_{\sigma,\tau_e,\mathsf{id}}\right|$*. Suppose that* iO *is an indistinguishability obfuscator against which every PPT adversary has advantage at most* $\mathsf{negl}(\lambda)/d(\lambda)$*. Then, no PPT adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *can win the game in Fig. 13 with non-negligible advantage.*

Proof. Let $\mathcal{A}$ be a PPT adversary. We proceed by means of $m + 1$ subhybrids indexed by $i = 0, 1, \ldots, m$. In the i-th of these hybrids, we provide $\mathcal{A}$ with an obfuscation of the program C_i' (see Fig. 16).

Observe that by the security of iO, when $i = 0$, Hybrid i is indistinguishable from the game in Fig. 13 when $b = 0$. Similarly, by the security of iO, when $i = m$, Hybrid i is indistinguishable from from the game in Fig. 13 when $b = 1$. It remains to prove that $\mathcal{A}$ cannot distinguish between Hybrid $i - 1$ and Hybrid i for any $i \in [m]$. We rely on Lemma 1.

We consider the circuit sampler Samp_i that runs SimSetup, provides σ and τ_e to $\mathcal{A}_1$, obtains $C, (\mathsf{id}_j)_{j \in [m]}, \psi$, compute τ_e^j for every $j \in [m]$ and outputs C_{i-1}', C_i', $\mathsf{aux} := \psi$ and $\rho := \bot$. Let $\mathcal{O}$ be an algorithm that always returns the empty diO Game for Almost Everywhere Extractable NIZKs

```
1. $b \xleftarrow{\$} \{0,1\}$
2. $(\sigma, \tau_s, \tau_e) \xleftarrow{\$} \mathsf{SimSetup}(\mathbb{1}^\lambda)$
3. $\big(C, (\mathsf{id}_j)_{j\in[m]}, \psi\big) \xleftarrow{\$} \mathcal{A}_1(\mathbb{1}^\lambda, \sigma, \tau_e)$
4. $\forall j \in [m]: \quad \tau_e^j \xleftarrow{\$} \mathsf{Trap}(\tau_e, \mathsf{id}_j)$
5. $\widetilde{C}_0 \xleftarrow{\$} \mathsf{iO}(\mathbb{1}^\lambda, C_0[\sigma, (\mathsf{id}_j)_{j\in[m]}])$ (see Fig. 14)
6. $\widetilde{C}_1 \xleftarrow{\$} \mathsf{iO}(\mathbb{1}^\lambda, C_1[\sigma, (\mathsf{id}_j)_{j\in[m]}, (\tau_e^j)_{j\in[m]}])$ (see Fig. 15)
7. The adversary wins if $\mathcal{A}_2(\psi, \widetilde{C}_b) = b$.
```

Fig. 13. diO game for almost everywhere extractable NIZKs

$C_0\big[\sigma, (\mathsf{id}_j)_{j\in[m]}\big]$

Hard-coded. The NIZK CRS σ, the m identities $(\mathsf{id}_j)_{j\in[m]}$.
Input. A set of inputs $(x_j)_{j\in[m]}$ and a set of proofs $(\pi_j)_{j\in[m]}$.

```
1. $\forall j \in [m]: \quad b_j \leftarrow \mathsf{NIZK.Verify}(\sigma, \mathsf{id}_j, \pi_j, x_j)$
2. If $\exists j \in [m]$ such that $b_j = 0$, output $\perp$
3. Output $C(x_1, \ldots, x_m)$
```

Fig. 14. The circuit C_0

string. We want to argue that even when aux is revealed, no PPT adversary can distinguish between the obfuscation of C'_{i-1} and C'_i.

Let $\ell(\lambda)$ and $d(\lambda)$ be the values used in the third and fourth property of our almost everywhere extractable NIZK. Let $\ell_0(\lambda)$ denote the position of the first bit of π_i. Define $\ell_1(\lambda) := \ell_0(\lambda) + \ell(\lambda)$.

The circuits C'_{i-1} and C'_i potentially have differing inputs. Observe that these must be values $(x_j, \pi_j)_{j\in[m]}$ for which $\mathsf{Verify}(\sigma, \mathsf{id}_i, \pi_i, x_i) = 1$ but $\mathsf{Extract}(\tau_e^i, \pi_i, x_i) = \perp$. In other words, we know that for every differing input,

$$\mathsf{DI}^{\ell_0,\ell_1}_{C'_{i-1},C'_i} \subseteq \mathsf{VPFE}_{\sigma,\tau_e,\mathsf{id}_i}.$$

With overwhelming probability over the randomness of $\mathsf{SimSetup}$, the latter has at most $d(\lambda)$ elements.

Now, suppose that there exists an adversary $\mathcal{B}$ running in time $\mathsf{poly}\big(\lambda, d(\lambda)\big)$ that can find an element in $\mathsf{DI}^{\ell_0,\ell_1}_{C'_{i-1},C'_i}$ with non-negligible probability given C'_{i-1}, C'_i and $\mathsf{aux} = \psi$. We build an adversary $\mathcal{B}'$ that breaks the fourth property of the almost everywhere extractable NIZK.

The adversary $\mathcal{B}'$ runs an internal copy of $\mathcal{A}_1$ and one of $\mathcal{B}$. It starts by providing the NIZK CRS σ and the trapdoor τ_e it received from its challenger to $\mathcal{A}_1$ obtaining C and $(\mathsf{id}_j)_{j\in[m]}$. Then, for every $j \in [m]$, $\mathcal{B}'$ computes $\tau_e^j \xleftarrow{\$}$

$C_1\left[\sigma, (\mathsf{id}_j)_{j\in[m]}, (\tau_e^j)_{j\in[m]}\right]$

Hard-coded. The NIZK CRS σ, the m identities $(\mathsf{id}_j)_{j\in[m]}$, the m extraction trapdoors $(\tau_e^j)_{j\in[m]}$.
Input. A set of inputs $(x_j)_{j\in[m]}$ and a set of proofs $(\pi_j)_{j\in[m]}$.

```
1. $\forall j \in [m]: \quad b_j \leftarrow \mathsf{NIZK.Verify}(\sigma, \mathsf{id}_j, \pi_j, x_j)$
2. $\forall j \in [m]: \quad w_j \leftarrow \mathsf{NIZK.Extract}(\tau_e^j, \pi_j, x_j)$
3. If $\exists j \in [m]$ such that $b_j = 0$ or $w_j = \perp$, output $\perp$
4. Output $C(x_1, \dots, x_m)$
```

Fig. 15. The circuit C_1

$C_i'\left[i, \sigma, (\mathsf{id}_j)_{j\in[m]}, (\tau_e^j)_{j\le i}\right]$

Hard-coded. The hybrid index i, the NIZK CRS σ where $(\sigma, \tau_s, \tau_e) \xleftarrow{\$} \mathsf{SimSetup}(\mathbb{1}^\lambda)$, the m identities $(\mathsf{id}_j)_{j\in[m]}$, the i extraction trapdoors $(\tau_e^j)_{j\le i}$ where $\tau_e^j \xleftarrow{\$} \mathsf{Trap}(\tau_e, \mathsf{id}_j)$ for every $j \le i$.
Input. A set of inputs $(x_j)_{j\in[m]}$ and a set of proofs $(\pi_j)_{j\in[m]}$.

```
1. $\forall j \le i: \quad w_j \leftarrow \mathsf{NIZK.Extract}(\tau_e^j, \pi_j, x_j)$
2. If $\exists j \le i$ such that $w_j = \perp$, output $\perp$
3. $\forall j \in [m]: \quad b_j \leftarrow \mathsf{NIZK.Verify}(\sigma, \mathsf{id}_j, \pi_j, x_j)$
4. If $\exists j \in [m]$ such that $b_j = 0$, output $\perp$
5. Output $C(x_1, \dots, x_m)$
```

Fig. 16. The circuit C_i'

$\mathsf{Trap}(\tau_e, \mathsf{id}_j)$. Finally, it provides $\mathcal{B}$ with C_{i-1}', C_i' and $\mathsf{aux} := \psi$ and outputs whatever $\mathcal{B}$ outputs. We observe that $\mathcal{B}'$ outputs an element in $\mathsf{DI}^{\ell_0,\ell_1}_{C_{i-1}',C_i'}$ with non-negligible probability. Furthermore, it runs in $\mathsf{poly}\big(\lambda, d(\lambda)\big)$ time.

We conclude that $\mathcal{A}_2$ cannot distinguish between the obfuscation of C_{i-1}' and C_i' even if it is given ψ. This ends the proof. □

Chosen-ID multi-theorem zero-knowledge. We now focus on formalising a particular zero-knowledge notion for almost everywhere extractable NIZKs. We call the property *chosen-ID* zero-knowledge. Informally, it says that, as long as τ_e^{id} remains secret, it is impossible to distinguish between a real proof under the identity id and a simulated one produced using the trapdoor τ_s.

This is formalised by giving the adversary access to an oracle that either generates real proofs using witnesses or simulates them using τ_s. We also give access to a second oracle that, on input any identity id, reveals the extraction trapdoor τ_e^{id}. The adversary is allowed to perform multiple adaptive queries to

Chosen-ID Zero-Knowledge Game

Initialisation: This procedure is run only once, at the beginning of the game.

1. $b \xleftarrow{\$} \{0,1\}$
2. $Q_0, Q_1 \leftarrow \emptyset$
3. $(\sigma, \tau_s, \tau_e) \xleftarrow{\$} \mathsf{SimSetup}(\mathbb{1}^\lambda)$
4. Activate the adversary with $\mathbb{1}^\lambda$ and σ.

Trapdoor: This procedure can be queried multiple times and at any point of the game. Upon receiving any query $(\mathsf{Trap}, \mathsf{id})$ where $\mathsf{id} \notin Q_1$, compute the following.

1. Add id to Q_0
2. $\tau_e^{\mathsf{id}} \xleftarrow{\$} \mathsf{Trap}(\tau_e, \mathsf{id})$
3. Give τ_e^{id} to the adversary.

Prove: This procedure can be queried multiple times and at any point of the game. Upon receiving any query $(\mathsf{Prove}, \mathsf{id}, x, w)$ where $\mathsf{id} \notin Q_0$ and $(x, w) \in \mathcal{R}$, compute the following.

1. Add id to Q_1
2. $\pi^0 \xleftarrow{\$} \mathsf{Prove}(\mathbb{1}^\lambda, \sigma, \mathsf{id}, x, w)$
3. $\pi^1 \xleftarrow{\$} \mathsf{SimProve}(\tau_s, \mathsf{id}, x)$
4. Give π^b to the adversary.

Win: The adversary wins if it guesses b.

Fig. 17. Chosen-ID zero-knowledge game

both the oracles with the only restriction that, if an identity is queried to the first oracle, it cannot be queried to the second one and vice-versa. Even with this kind of help, the adversary should not be able to tell if it is given real proofs or fake ones.

Definition 18 (Chosen-ID Zero-knowledge NIZK). *An almost everywhere extractable NIZK* $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ *for* $\mathcal{R}$ *is chosen-ID zero-knowledge if there exists a uniform PPT algorithm* $\mathsf{SimProve}$ *such that no PPT adversary* $\mathcal{A}$ *can win the game in Fig. 17 with non-negligible advantage.*

4.1 Building almost everywhere extractable NIZKs

We explain how to build a chosen-ID zero-knowledge, almost everywhere extractable NIZK with security against non-uniform adversaries.

Our construction relies on an identity based encryption scheme, a non-interactive commitment scheme, a subexponentially secure injective one-way function and a NIWI proof. The CRS will consist of the IBE master public key and a challenge v for a one-way function. It is possible to instantiate the primitives so that the CRS is short (i.e. the length depends only on the security parameter) and unstructured.

Let $\mathcal{R}$ be the NP relation we are targetting, suppose that we want to prove that $x \in L_{\mathcal{R}}$ using the identity id. The proof is obtained by encrypting the witness w under the identity id using the IBE scheme. We also commit to 0. Then, we generate a NIWI proving that either the ciphertext is an encryption of w under id, or we committed to the preimage of v. The NIZK proof consists of the concatenation of the commitment, the ciphertext and the NIWI. The verification is a simple check of the latter.

Observe that it is easy to extract the witness by decrypting the ciphertext. Of course, the operation requires knowing the private key associated with the identity id. The latter can be derived from the master secret key of the IBE scheme. Even simulating proofs is rather easy: it is sufficient to encrypt 0, commit to a preimage of v and use the latter as witness for the NIWI. To summarise, the extraction trapdoor will be the master secret key, the soundness trapdoor will be the preimage of v.

Ensuring almost-everywhere extractability. Our idea is that, in all proofs where the witness cannot be extracted, the commitment will hide a preimage of v. In order to ensure this, we will rely on a perfectly correct IBE scheme (if the ciphertext hides the witness, we always succeed in extracting it) and a perfectly sound NIWI (if the ciphertext does not hide a witness, the commitment must hide a preimage of v). Since the one-way function is injective, there will be at most $2^{q(\lambda)}$ ways of committing to a preimage of v. Here, $q(\lambda)$ denotes the length of the randomness used by the commitment[10].

Now, suppose that the commitment is perfectly binding and it is possible to break hiding in $\mathsf{poly}\big(\lambda, 2^{q(\lambda)}\big)$ time. By choosing a sufficiently large security parameter for the one-way function, we can make sure that finding the preimage of v is hard even for adversaries running in $\mathsf{poly}\big(\lambda, 2^{q(\lambda)}\big)$ time. That ensures the last property of almost everywhere extractable NIZKs.

Proving chosen-ID zero-knowledge is instead rather easy. We just rely on witness-indistinguishability, the hiding properties of the commitment and the IND-ID-CPA security of IBE. Notice that a message encrypted under the identity id remains secret as long as the secret-key for id is kept private. Leaking private keys for other identities does not help in retrieving the plaintext.

Formalising the construction. Let $\mathcal{R}$ the NP relation for our almost everywhere extractable NIZK. Consider an IND-ID-CPA identity-based encryption scheme IBE where the master public key mpk is computationally indistinguishable from a uniformly random string. We also require that the scheme satisfies perfect correctness. For instance, we can use the constructions of [BB04,ABB10].

Let Com be a computationally hiding, perfectly binding non-interactive commitment scheme without CRS. Suppose that there exists an algorithm running in superpolynomial time that breaks hiding with probability 1. Finally, let OWF be

[10] We can assume that $q(\lambda)$ is independent of the length of the committed value. Consider for instance a scheme in which we commit the message bit by bit and all the randomness comes from a PRF.

a subexponentially secure injective one-way function. Furthermore, assume that the one-way function outputs values that are computationally indistinguishable from a uniformly random string. This kind of one-way function can be instantiated e.g. using DLOG.

Finally, we rely on a NIWI scheme without CRS. The underlying relation is the following.

$$\mathcal{R}_{\mathsf{NIWI}} := \left\{ \begin{array}{c|c} \big((\mathsf{mpk}, v, \mathsf{id}, c_0, c_1, x), & c_1 = \mathsf{Enc}(\mathsf{mpk}, \mathsf{id}, w;\, r), \quad (x, w) \in \mathcal{R} \\ (w, r)\big) & \text{OR} \\ & w = u, \quad c_0 = \mathsf{Com}(u;\, r), \quad \mathsf{OWF}(\mathbb{1}^\lambda, u) = v \end{array} \right\}$$

Our construction is formalised in Fig. 18 and Fig. 19.

```
CHOSEN-ID ZERO-KNOWLEDGE, ALMOST EVERYWHERE EXTRACTABLE NIZK - PART 1
Let $q_1(\lambda)$ denote the length of the randomness needed by $\mathsf{IBE.Enc}$.
$\mathsf{Setup}(\mathbb{1}^\lambda)$

  1. $(\mathsf{mpk}, \mathsf{msk}) \xleftarrow{\$} \mathsf{IBE.Setup}(\mathbb{1}^\lambda)$
  2. $(v, u) \xleftarrow{\$} \mathsf{OWF.Gen}(\mathbb{1}^\lambda)$
  3. Output $\sigma := (\mathsf{mpk}, v)$

$\mathsf{Prove}\big(\mathbb{1}^\lambda, \sigma = (\mathsf{mpk}, v), \mathsf{id}, x, w\big)$

  1. $c_0 \xleftarrow{\$} \mathsf{Com}(\mathbb{1}^\lambda, 0)$
  2. $r \xleftarrow{\$} \{0,1\}^{q_1(\lambda)}$
  3. $c_1 \leftarrow \mathsf{IBE.Enc}(\mathsf{mpk}, \mathsf{id}, w;\, r)$
  4. $\pi' \xleftarrow{\$} \mathsf{NIWI.Prove}\big(\mathbb{1}^\lambda, (\mathsf{mpk}, v, \mathsf{id}, c_0, c_1, x), (w, r)\big)$
  5. Output $\pi := (c_0, c_1, \pi')$

$\mathsf{Verify}\big(\sigma = (\mathsf{mpk}, v), \mathsf{id}, \pi = (c_0, c_1, \pi'), x\big)$

  1. Output $\mathsf{NIWI.Verify}\big(\pi', (\mathsf{mpk}, v, \mathsf{id}, c_0, c_1, x)\big)$
```

Fig. 18. A chosen-ID zero-knowledge, almost everywhere extractable NIZK - Part 1

Theorem 12. *Suppose that* Com *is a computationally hiding, perfectly binding non-interactive commitment. Assume that the algorithm needs* $q_2(\lambda)$ *bits of randomness. Suppose that there exists an algorithm running in* $\mathsf{poly}\big(\lambda, S(\lambda)\big)$ *time that breaks the hiding property of* Com *with probability* 1.

Let IBE *be an IND-ID-CPA identity-based encryption scheme that satisfies perfect correctness. Let* OWF *be an injective one-way function that is hard to invert even for adversaries running in* $\mathsf{poly}\big(\lambda, 2^{q_2(\lambda)}, S(\lambda)\big)$ *time. Suppose that* NIWI *is a perfectly sound witness-indistinguishable proof system for the relation* $\mathcal{R}_{\mathsf{NIWI}}$.

```
Chosen-ID Zero-Knowledge, Almost Everywhere Extractable NIZK - Part 2
Let $q_2(\lambda)$ denote the length of the randomness needed by $\mathsf{Com}$.
$\mathsf{SimSetup}(\mathbb{1}^\lambda)$
  1. $(\mathsf{mpk}, \mathsf{msk}) \xleftarrow{\$} \mathsf{IBE.Setup}(\mathbb{1}^\lambda)$
  2. $(v, u) \xleftarrow{\$} \mathsf{OWF.Gen}(\mathbb{1}^\lambda)$
  3. Output $\sigma := (\mathsf{mpk}, v), \tau_s := u, \tau_e := \mathsf{msk}$

$\mathsf{SimProve}(\tau_s = u, \mathsf{id}, x)$
  1. $r \xleftarrow{\$} \{0,1\}^{q_2(\lambda)}$
  2. $c_0 \leftarrow \mathsf{Com}(\mathbb{1}^\lambda, u;\, r)$
  3. $c_1 \xleftarrow{\$} \mathsf{IBE.Enc}(\mathsf{mpk}, \mathsf{id}, 0)$
  4. $\pi' \xleftarrow{\$} \mathsf{NIWI.Prove}\big(\mathbb{1}^\lambda, (\mathsf{mpk}, v, \mathsf{id}, c_0, c_1, x), (u, r)\big)$
  5. Output $\pi := (c_0, c_1, \pi')$

$\mathsf{Trap}(\tau_e = \mathsf{msk}, \mathsf{id})$
  1. Output $\mathsf{IBE.Extract}(\mathsf{msk}, \mathsf{id})$

$\mathsf{Extract}(\tau_e^{\mathsf{id}}, \pi = (c_0, c_1, \pi'), x)$
  1. $w \leftarrow \mathsf{IBE.Dec}(\tau_e^{\mathsf{id}}, c_1)$
  2. If $(x, w) \in \mathcal{R}$, output $w$, otherwise, output $\perp$.
```

Fig. 19. A chosen-ID zero-knowledge, almost everywhere extractable NIZK - Part 2

Then, the construction in Fig. 18 and Fig. 19 is a chosen-ID zero-knowledge almost everywhere extractable NIZK for $\mathcal{R}$ against non-uniform PPT adversaries.

We prove Theorem 12 in Appendix A.

5 Weakening Distributed Samplers to Avoid Random Oracles

In this section, we reformulate the concept of distributed sampler under a new light. Although we weaken the simulation-based definition of [ASY22], we obtain a meaningful notion of security against active adversaries. This allows us to build constructions that overcome the impossibilities of [AOS23] without using random oracles.

Syntax of Distributed Samplers. We start by recalling the syntax of distributed samplers [ASY22].

Definition 19 (Distributed Sampler). *An n-party distributed sampler is a triple of uniform, PPT algorithms* $(\mathsf{Setup}, \mathsf{Gen}, \mathsf{Sample})$ *with the following syntax:*

- Setup *is a probabilistic algorithm taking as input the security parameter. The output is a string* crs.
- Gen *is a probabilistic algorithm taking as input the security parameter, a session identity* sid*, the index $i \in [n]$ of the party running the algorithm and the string* crs*. The output is the distributed sampler message U_i of the i-th party.*
- Sample *is a deterministic algorithm taking as input n distributed sampler messages $U_1, U_2, \ldots, U_n$, a session identity* sid *and the string* crs*. The output is a sample R.*

Observe that distributed samplers are implicitly associated with a one-round protocol with CRS (the latter is generated using $\mathsf{Setup}(\mathbb{1}^\lambda)$) producing a sample from a target distribution $\mathcal{D}$. In such protocol, all the parties P_i simultaneously broadcast a distributed sampler message $U_i \overset{\$}{\leftarrow} \mathsf{Gen}(\mathbb{1}^\lambda, \mathsf{sid}, i, \mathsf{crs})$. After that, everybody retrieves the output $R \leftarrow \mathsf{Sample}(U_1, U_2, \ldots, U_n, \mathsf{sid}, \mathsf{crs})$.

Notice that, compared to [ASY22], we augmented the generation and sampling algorithms with a session identity. The latter can be used to restrict the context in which the distributed sampler messages can be used. For instance, it can identify the identities of the parties taking part to the protocol. If the session identity of any of the exchanged messages does not match the expected set of parties, the sampling algorithm will produce $\perp$.

5.1 Hardness-Preserving Distributed Samplers.

We now present the first weakening of the original definition. The notion is called *hardness-preserving distributed sampler.* The name refers to the fact that this kind of distributed sampler allows compiling protocols with CRS Π into protocols without CRS Π' while preserving the hardness properties: if the probability of realising an attack against Π is negligible, the probability of realising the same attack against Π' still remains negligible.

An unusual definition of security. Our definition is based on a real-world/ideal-world paradigm where simulation is non-black-box. In the real world, the adversary is provided with a distributed sampler CRS and the message of a honest party. After selecting the distributed sampler messages of the other parties, the adversary is provided with the output of the protocol (notice that the adversary was already able to compute this on its own). In the ideal world, instead, the CRS and the message of the honest party are produced by a simulator. The latter is given an ideal sample $R \overset{\$}{\leftarrow} \mathcal{D}(\mathbb{1}^\lambda)$. When the adversary answers with the distributed sampler messages of the other players, we do not compute the output of the protocol, we just provide the adversary with R.

The important point is that we do not ask for indistinguishability between the real-world and the ideal world. That would indeed be impossible to achieve.

```
THE HARDNESS-PRESERVING GAME $\mathcal{G}_{\mathsf{HP}}$
Each phase is run only once.
Initialisation Phase:
  1. $b \xleftarrow{\$} \{0,1\}$
  2. $R^{(1)} \xleftarrow{\$} \mathcal{D}(\mathbb{1}^\lambda)$
  3. $\mathsf{crs}^{(0)} \xleftarrow{\$} \mathsf{Setup}(\mathbb{1}^\lambda)$
  4. $(\mathsf{crs}^{(1)}, \zeta) \xleftarrow{\$} \mathsf{SimSetup}_{\mathcal{A}}(\mathbb{1}^\lambda)$
  5. Activate $\mathcal{A}$ with $\mathbb{1}^\lambda$ and $\mathsf{crs}^{(b)}$.
Generation Phase:
  1. Receive $i \in [n]$ and a session identity $\mathsf{sid} = (\mathsf{tag}, \mathsf{id}_{j_1}, \dots, \mathsf{id}_{j_n})$ from $\mathcal{A}$
  2. $U^{(0)} \xleftarrow{\$} \mathsf{Gen}(\mathbb{1}^\lambda, \mathsf{sid}, i, \mathsf{crs}^{(0)})$
  3. $U^{(1)} \xleftarrow{\$} \mathsf{SimGen}_{\mathcal{A}}(\mathbb{1}^\lambda, \mathsf{sid}, i, \zeta, R^{(1)})$
  4. $U_i \leftarrow U^{(b)}$
  5. Provide $U_i$ to $\mathcal{A}$
Sampling Phase
  1. Receive $(U_j)_{j \neq i}$ from $\mathcal{A}$
  2. $R^{(0)} \leftarrow \mathsf{Sample}(U_1, \dots, U_n, \mathsf{crs}^{(0)})$
  3. If $R^{(0)} = \bot$, output 0.
  4. Otherwise, provide $R := R^{(b)}$ to $\mathcal{A}$
  5. The output of the game is the bit output by $\mathcal{A}$.
```

Fig. 20. The hardness-preserving game $\mathcal{G}_{\mathsf{HP}}$

We ask instead that if an adversary $\mathcal{A}$ outputs 1 with non-negligible probability while interacting with the real world, then, $\mathcal{A}$ outputs 1 with non-negligible probability even while interacting with the ideal world.

Definition 20 (Hardness-Preserving Distributed Sampler). *Let $\mathcal{D}(\mathbb{1}^\lambda)$ be an efficient distribution. We say that an n-party distributed sampler is hardness-preserving for $\mathcal{D}(\mathbb{1}^\lambda)$ against AClass if, for every PPT $\mathcal{A} \in \mathsf{AClass}$, there exists a pair of PPT non-uniform simulators $(\mathsf{SimSetup}_{\mathcal{A}}, \mathsf{SimGen}_{\mathcal{A}})$ such that, in the game $\mathcal{G}_{\mathsf{HP}}$ in Fig. 20,*

$$\Pr\left[\mathcal{G}_{\mathsf{HP}}^{\mathcal{A}}(\mathbb{1}^\lambda) = 1 \middle| b = 0\right] = \mathsf{nonegl}(\lambda) \implies \Pr\left[\mathcal{G}_{\mathsf{HP}}^{\mathcal{A}}(\mathbb{1}^\lambda) = 1 \middle| b = 1\right] = \mathsf{nonegl}(\lambda).$$

Preservation of hardness. We now explain in what sense distributed samplers satisfying Def. 20 preserve hardness.

We start by formalising the concept of *game with oracle distribution*. This basically corresponds to a game describing the interaction between n parties connected by authenticated point-to-point channels and a broadcast medium.

```
GAME WITH ORACLE DISTRIBUTION

 1. Activate the adversary $\mathcal{A}$ with $\mathbb{1}^\lambda$.
 2. Receive $\mathsf{aux}$, $H \subseteq [n]$ from $\mathcal{A}$.
 3. $R \xleftarrow{\$} \mathcal{D}(\mathbb{1}^\lambda)$
 4. Activate a copy of $\mathsf{Ch}$ with $\mathbb{1}^\lambda$, $H$ and $\mathsf{aux}$.
 5. Relay all the messages from $\mathsf{Ch}$ to $\mathcal{A}$.
 6. Relay all the messages from $\mathcal{A}$ to $\mathsf{Ch}$.
 7. After the challenger has sent $(\mathsf{Sample}, j)$ for every $j \in H$, provide $\mathcal{A}$ with $R$.
 8. Only when the above occurred, after $\mathcal{A}$ has sent $(\mathsf{Sample}, j)$ for every $j \notin H$,
    provide $R$ to $\mathsf{Ch}$.
 9. Keep relaying the messages between $\mathcal{A}$ and $\mathsf{Ch}$ as before.
10. The output of the game is the value output by $\mathsf{Ch}$ before halting.
```

Fig. 21. Game with oracle distribution

The adversary has full control over the corrupted players, whereas the operations of the honest parties is managed by the challenger of the game. The novelty compared to the a standard game is that, at some point in time, the parties are all provided with the same ideal sample R from a distribution $\mathcal{D}(\mathbb{1}^\lambda)$. The moment in which the sample is delivered is chosen by the parties themselves: by sending a special message (Sample, i), the i-th party declares its approval on delivering R. When all the honest parties expressed their agreement, the sample R is provided to the adversary. When all the corrupted parties agree too, the sample R is given to the challenger too. The adversary wins the game if the challenger terminates its execution outputting 1. For instance, this can mean that the adversary succeeded in performing an attack. We define the advantage as the probability of this event.

Definition 21 (Game with oracle distribution). *An n-party game with oracle distribution is a triple $\mathcal{G} := (\mathcal{D}, \mathsf{Ch})$ where*

1. *$\mathcal{D}(\mathbb{1}^\lambda)$ is an efficient distribution: a uniform, PPT algorithm taking only the security parameter as input.*
2. *Ch is an efficient challenger: a uniform, PPT, round-based, interactive Turing machine that, for every $i \in [n]$, sends the message (Sample, i) at most once in its execution.*

Let $\mathcal{A}$ be a round-based interactive Turing machine. We define $\mathcal{G}_{\mathcal{A}}(\mathbb{1}^\lambda)$ to be the output of the game in Fig. 21.

For every adversary $\mathcal{A}$, we define the advantage of $\mathcal{A}$ in the game $\mathcal{G}$ as

$$\mathsf{Adv}^{\mathcal{G}}_{\mathcal{A}}(\lambda) := \Pr\Big[\mathcal{G}_{\mathcal{A}}(\mathbb{1}^\lambda) = 1\Big].$$

We say that $\mathcal{A}$ wins with non-negligible advantage if $\mathsf{Adv}^{\mathcal{G}}_{\mathcal{A}}(\lambda)$ is non-negligible in the security parameter.

Notice that at the beginning of the game in Def. 21, the adversary is allowed to choose the set of honest parties H and an auxiliary input aux for Ch. In other words, our definition considers only static corruption in the dishonest majority setting.

On the expressiveness of the model. Protocols relying on CRSs can be formulated as games with oracle distribution. In such settings, $\mathcal{D}(\mathbb{1}^\lambda)$ represents the distribution from which the CRS is generated. Since the CRS should be given before the beginning of the protocol, in the corresponding game with oracle distribution, the challenger immediately starts by sending (Sample, i) for every $i \in H$. It then waits for analogous messages from the corrupted players, ignoring all other communication. After that, the challenger runs the protocol with the adversary on behalf of the honest parties.

More in general, games with oracle distributions can be used to analyse the security of protocols that rely on a sampling resource: a functionality that, upon receiving the approval of all players, delivers an ideal sample from a fixed distribution $\mathcal{D}(\mathbb{1}^\lambda)$. The sample is leaked to the adversary in advance, at the moment in which all the honest players send their approval.

Compiling games with oracle distributions using distributed samplers. Using a distributed sampler for $\mathcal{D}$, there is a natural way to compile a game with oracle distribution $\mathcal{G} = (\mathcal{D}, \mathsf{Ch})$ into a standard interactive game. The delivery of a special message (Sample, i) in $\mathcal{G}$ will correspond to the delivery of a distributed sampler message U_i from party P_i. The sample R used by the challenger Ch will be the output of the distributed sampler. If the output is $R = \perp$, the challenger always halts outputting 0.

Observe that, as in the game with oracle distribution, the adversary can learn the sample R as soon as all the honest parties deliver their distributed sampler messages. Indeed, the adversary may have already chosen the distributed sampler messages of the corrupted players without revealing them. The honest players (i.e. the challenger) will discover R only when the adversary decides to deliver these messages.

Notice that in the case of a protocol with CRS Π, the compiled game consists of the sequential composition of a distributed sampler with Π, where the former is used to generate the CRS for the latter.

Multi-session security. Distributed samplers sometimes make use of a CRS. We would like the latter to be reusable among multiple sessions involving different subsets of parties. Since all these executions are correlated by the use of the same CRS, the security analysis of the compiled game cannot restrict to single sessions. For this reason, upon activation, we provide the adversary with the distributed sampler CRS and we let it choose the identities of a large number $m > n$ of players that will constitute our universe. At the same time, it also selects the set of honest players H. At that point, the adversary is free to engage in many, possibly simultaneous sessions of the compiled game, all using the same distributed sampler CRS. Each session takes place between n parties chosen by

Compiled Game with Oracle Distribution

Initialisation: This procedure is run only once, at the beginning of the game.

1. $\mathsf{crs} \overset{\$}{\leftarrow} \mathsf{DS.Setup}(\mathbb{1}^\lambda)$
2. Activate the adversary $\mathcal{A}$ with $\mathbb{1}^\lambda$ and crs.
3. Receive a list of identities of the parties $\mathsf{ID} := \{\mathsf{id}_1, \ldots, \mathsf{id}_m\}$ from $\mathcal{A}$ along with the subset of honest players $H \subseteq [m]$.

Session: This procedure can be queried multiple times and at any point of the game. Upon receiving any query $(\mathsf{NewSession}, \mathsf{tag}, \mathsf{id}_{j_1}, \ldots, \mathsf{id}_{j_n}, \mathsf{aux})$ where the session identity $\mathsf{sid} := (\mathsf{tag}, \mathsf{id}_{j_1}, \ldots, \mathsf{id}_{j_n})$ has not been queried before, $\mathsf{id}_{j_i} \in \mathsf{ID}$ for every $i \in [n]$, $\mathsf{id}_{j_l} \neq \mathsf{id}_{j_k}$ for every $l \neq k$ and aux, perform the following.

1. Store sid
2. $\forall i \in [n]$ such that $j_i \in H$: $\quad U_i \overset{\$}{\leftarrow} \mathsf{DS.Gen}(\mathbb{1}^\lambda, \mathsf{sid}, i, \mathsf{crs})$
3. Activate a new copy of Ch with $\mathbb{1}^\lambda$, $H' := \{i \in [n] | j_i \in H\}$ and aux.
4. Relay all the messages from Ch to $\mathcal{A}$ appending sid to them.
5. Relay all the messages from $\mathcal{A}$ with prefix sid to Ch (the prefix is removed).
6. When Ch sends (Sample, i) for any $i \in H'$, provide $\mathcal{A}$ with $(\mathsf{sid}, \mathsf{Sample}, \mathsf{id}_{j_i}, U_i)$.
7. When $\mathcal{A}$ sends $(\mathsf{sid}, \mathsf{Sample}, \mathsf{id}_{j_i}, U_i)$ for any $i \notin H'$, give (Sample, i) to Ch.
8. When all the messages $(\mathsf{sid}, \mathsf{Sample}, \mathsf{id}_{j_i}, U_i)_{i \in [n]}$ have been exchanged, provide Ch with $R \leftarrow \mathsf{DS.Sample}(U_1, \ldots, U_n, \mathsf{sid}, \mathsf{crs})$.
9. Keep relaying the messages between Ch and $\mathcal{A}$ as before.
10. The output of the session is the value output by Ch before halting. If $R = \bot$, the output of the session is 0.

Output: In the game, multiple sessions are run in parallel. The output of the game is 1 if there exists a session that terminates with 1.

Fig. 22. Compiled game with oracle distribution

the adversary. The session is uniquely identified by a session label sid consisting of the identities of the n parties and an additional label tag that acts like a counter. Thanks to the latter, it will be possible to have multiple sessions among the same subset of parties. For each session, the adversary is also allowed to choose a different auxiliary input aux. We define the advantage of the adversary as the probability that, in one of the sessions, the challenger outputs 1.

Definition 22 (Compiled game). *Let $\mathcal{G} = (\mathcal{D}, \mathsf{Ch})$ be an n-party game with oracle distribution and let $\mathsf{DS} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Sample})$ be an n-party distributed sampler. We define the compiled game $\mathcal{G}'$ in Fig. 22. For any PPT adversary $\mathcal{A}$, we denote the output of the game by $\mathcal{G}'_{\mathcal{A}}(\mathbb{1}^\lambda)$. We denote the value output by $\mathcal{A}$ before halting by $\mathcal{A}_{\mathcal{G}'}(\mathbb{1}^\lambda)$.*

We define the advantage of $\mathcal{A}$ in the game $\mathcal{G}'$ as

$$\mathsf{Adv}^{\mathcal{G}'}_{\mathcal{A}}(\lambda) := \Pr\Big[\mathcal{G}'_{\mathcal{A}}(\mathbb{1}^\lambda) = 1\Big].$$

We say that $\mathcal{A}$ wins with non-negligible advantage if $\mathsf{Adv}^{\mathcal{G}'}_{\mathcal{A}}(\lambda)$ is non-negligible in the security parameter.

In the next theorem, we show that if the distributed sampler is hardness-preserving, hard-to-win games with oracle distribution are compiled into standard games that are still hard to win. In other words, if we have a protocol with CRS Π for which all PPT adversaries fail in performing an attack, the attack remains hard to perform even against the compiled protocol Π'.

Theorem 13. *Let $\mathcal{G} := (\mathcal{D}, \mathsf{Ch})$ be an n-party game with oracle distribution such that every PPT adversary $\mathcal{A}$ has negligible advantage against $\mathcal{G}$. Let $\mathsf{DS} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Sample})$ be an n-party distributed sampler. If DS is hardness-preserving for $\mathcal{D}$ against AClass, there exists no PPT $\mathcal{A}' \in \mathsf{AClass}$ such that $\mathsf{Adv}^{\mathcal{G}'}_{\mathcal{A}'}(\lambda)$ is non-negligible.*

The idea at the base of the proof is rather simple. Suppose that an adversary $\mathcal{A}'$ can win against the compiled game with non-negligible advantage. That means that, if we pick a session at random, the session output is 1 with non-negligible probability. Now, we build an the adversary $\mathcal{B}$ against the hardness-preserving property of the distributed sampler. The latter picks a random session ι of the compiled game and simulates it to $\mathcal{A}'$ using the values provided by its challenger. In particular, $\mathcal{B}$ is given the CRS crs, the honest distributed sampler message that is sent for last and the distributed sampler output. The adversary $\mathcal{B}$ halts outputting the outcome of the ι-th session.

In the real-world execution of the distributed sampler, $\mathcal{B}$ outputs 1 with non-negligible probability, so, by the hardness-preserving properties, the same must happen in the ideal-world execution. In the latter, however, in $\mathcal{B}$'s simulation of the ι-th session, the challenger is given an ideal sample from $\mathcal{D}(\mathbb{1}^\lambda)$ instead of the actual distributed sampler output. From this, we can easily build a PPT adversary $\mathcal{A}$ that wins against $\mathcal{G}$ with non-negligible advantage.

Proof. Suppose that our game is false and there exists a PPT adversary $\mathcal{A}' \in \mathsf{AClass}$ such that $\mathsf{Adv}^{\mathcal{G}'}_{\mathcal{A}'}(\lambda)$ is non-negligible. Let $M(\lambda)$ be a polynomial upper-bounding the number of $\mathsf{NewSession}$ queries issued by $\mathcal{A}$.

We construct a PPT adversary $\mathcal{B} \in \mathsf{AClass}$ for the hardness-preserving game such that

$$\Pr\left[\mathcal{G}^{\mathcal{B}}_{\mathsf{HP}}(\mathbb{1}^\lambda) = 1 \middle| b = 0\right] = \mathsf{nonegl}(\lambda). \qquad (1)$$

The adversary $\mathcal{B}$ starts its execution by selecting a random value $\iota \xleftarrow{\$} [M]$. Then, it uses the value crs given by its challenger to simulate $\mathcal{G}'$ to an internal copy of $\mathcal{A}'$. It behaves slightly differently in the ι-th $\mathsf{NewSession}$ query. Specifically, let (Sample, i) be the last special message sent by Ch in that session. Instead of providing a distributed sampler message generated using $\mathsf{DS.Gen}$, the adversary $\mathcal{B}$ queries its challenger with i and $\mathsf{sid} = (\mathsf{tag}, \mathsf{id}_{j_1}, \dots, \mathsf{id}_{j_n})$. It provides the adversary with the answer U_i. Moreover, after all the distributed sampler messages $(U_j)_{j\in[n]}$ have been exchanged, $\mathcal{B}$ does not compute the sample R using $\mathsf{DS.Sample}$, but queries its challenger with $(U_j)_{j\neq i}$. It gives the answer to Ch. All the rest remains as in Fig. 22. The final output of $\mathcal{B}$ corresponds to the output of the ι-th session.

We observe that if the bit b in the hardness-preserving game is set to 0, the view of $\mathcal{A}'$ in $\mathcal{G}'$ coincides with the one in $\mathcal{B}$'s simulation. So,

$$\Pr\left[\mathcal{G}^{\mathcal{B}}_{\mathsf{HP}}(\mathbb{1}^\lambda) = 1 \middle| b = 0\right] \geq \frac{1}{M(\lambda)} \cdot \Pr\left[\mathcal{G}'_{\mathcal{A}'}(\mathbb{1}^\lambda) = 1\right].$$

The latter is non-negligible. Notice also that since the challenger of $\mathcal{G}'$ is uniform and PPT, $\mathcal{B}$ still belongs to AClass. We have just proven equation (1).

By the hardness-preserving property of DS, we know that there exists a pair of PPT algorithms $(\mathsf{SimSetup}_{\mathcal{B}}, \mathsf{SimGen}_{\mathcal{B}})$ such that

$$\Pr\left[\mathcal{G}^{\mathcal{B}}_{\mathsf{HP}}(\mathbb{1}^\lambda) = 1 \middle| b = 1\right] = \mathsf{nonegl}(\lambda). \tag{2}$$

We can finally build a PPT adversary $\mathcal{A}$ that wins the game $\mathcal{G}$ with non-negligible advantage. The adversary $\mathcal{A}$ runs an internal copy of $\mathcal{A}'$. It starts its execution by sampling $\iota \overset{\$}{\leftarrow} [M]$ and running $(\mathsf{crs}, \zeta) \overset{\$}{\leftarrow} \mathsf{SimSetup}_{\mathcal{B}}(\mathbb{1}^\lambda)$. Then, it simulates the game $\mathcal{G}'$ to $\mathcal{A}'$ using crs as CRS for the distributed sampler. The simulation of the game takes place as in Fig. 22 with the exception of the ι-th session. Let $\mathsf{sid} = (\mathsf{tag}, \mathsf{id}_{j_1}, \dots, \mathsf{id}_{j_n})$ be the corresponding session identity and aux the corresponding auxiliary input. The adversary $\mathcal{A}$ provides its challenger with aux and the set of honest players $\{i \in [n] | j_i \in H\}$. Then, it relays the messages between $\mathcal{A}'$ and Ch. When $\mathcal{A}$ receives (Sample, i) where $i \in H$ from its challenger, it generates a distributed sampler message U_i and sends it to $\mathcal{A}'$. The operations is always performed using $\mathsf{DS.Gen}$ except for the last honest player. In that case, $\mathcal{A}$ receives an ideal sample R from its challenger, so, it generates U_i using

$$U_i \overset{\$}{\leftarrow} \mathsf{DS.SimGen}_{\mathcal{B}}(\mathbb{1}^\lambda, \mathsf{sid}, i, \zeta, R).$$

When $\mathcal{A}'$ sends a distributed sample message in the ι-th session on behalf of a corrupted party id_{j_i}, $\mathcal{A}$ sends (Sample, i) to Ch. The adversary $\mathcal{A}$ terminates its execution when $\mathcal{A}'$ does.

We observe that

$$\Pr\left[\mathcal{G}_{\mathcal{A}}(\mathbb{1}^\lambda) = 1\right] = \Pr\left[\mathcal{G}^{\mathcal{B}}_{\mathsf{HP}}(\mathbb{1}^\lambda) = 1 \middle| b = 1\right] = \mathsf{nonegl}(\lambda).$$

□

5.2 Indistinguishability Preserving Distributed Samplers

Hardness-preserving distributed samplers guarantee a somewhat limited form a security: they are just meant to preserve the hardness of computations. In other words, if we have two indistinguishable games relying on a CRS, a hardness-preserving distributed sampler does not guarantee that the compiled games are still indistinguishable.

More concretely, suppose that we deal with the security proof of a protocol Π relying on a CRS R. That means that there exists a simulator $\mathcal{S}$ such that Π is indistinguishable from the interaction between $\mathcal{S}$ and a functionality $\mathcal{F}$. A hardness-preserving distributed sampler does not guarantee that the compiled protocol Π' still implements the functionality $\mathcal{F}$. Indeed, how can we simulate the distributed sampler messages sent in Π'? Notice that in its simulation, $\mathcal{S}$ might rely on a trapdoored version of the CRS R. It can be that the outputs of the hardness-preserving distributed sampler never have a trapdoor. Furthermore, even if the trapdoor existed, how would $\mathcal{S}$ retrieve it?

We need our distributed sampler to satisfy additional properties. For this reason, we introduce the notion of *indistinguishability-preserving distributed sampler*. They will guarantee that, under some conditions, if a protocol Π relying on a CRS implements a functionality $\mathcal{F}$ against an active adversary in the UC model, the compiled protocol still implements $\mathcal{F}$. As for the hardness-preserving case, indistinguishability-preserving distributed samplers overcome the impossibilities of [AOS23]. They can therefore be built without using random oracles.

Roadmap for the definition. In order to formalise the definition of indistinguishability-preserving distributed sampler, we need to introduce preliminary concepts. We will define a trapdoored version of games with oracle distribution. This notion is meant to model the behaviour of a simulator that hides trapdoors in the CRSs it produces. In a game with trapdoor oracle distribution, the ideal sample given to the parties hides a trapdoor T. The latter is revealed only to the challenger simultaneously with R. We then define indistinguishability between a game with oracle distribution and a game with trapdoored oracle distribution. Finally, we define indistinguishability-preserving distributed samplers as distributed samplers that compile games with oracle distribution and games with trapdoored oracle distribution preserving indistinguishability.

Games with trapdoored oracle distribution. We introduce the concept of *trapdoored distribution*. Essentially, the latter consists of a distribution $\mathcal{D}'$ that outputs samples R along with trapdoors T. The trapdoor distribution $\mathcal{D}'$ can also be given an auxiliary input aux' of fixed length. The notion is formalised with respect to another (standard) distribution $\mathcal{D}$. We require that for every value aux', the sample R generated by $\mathcal{D}'$ is indistinguishable from the one generated by $\mathcal{D}$.

Definition 23 (Trapdoored distribution). *Let $\mathcal{D}(\mathbb{1}^\lambda)$ be an efficient distribution. A trapdoored distribution for $\mathcal{D}$ is a uniform, PPT algorithm $\mathcal{D}'$ which takes as input the security parameter $\mathbb{1}^\lambda$ and auxiliary information $\mathsf{aux}' \in \{0,1\}^{\ell(\lambda)}$ where $\ell(\lambda)$ is a fixed polynomial. The outputs are a sample R and a trapdoor T. We also require that, for every auxiliary input $\mathsf{aux}' \in \{0,1\}^{\ell(\lambda)}$, the following distributions are indistinguishable*

$$\left\{R \,\middle|\, R \xleftarrow{\$} \mathcal{D}(\mathbb{1}^\lambda)\right\} \qquad \text{and} \qquad \left\{R \,\middle|\, (R,T) \xleftarrow{\$} \mathcal{D}'(\mathbb{1}^\lambda, \mathsf{aux}')\right\}.$$

Trapdoored distributions are meant to represent the distributions used by simulators of MPC protocols. The auxiliary input aux' can be used to represent any information that the simulator receives from the functionality such as public inputs. It may happen indeed that the simulated CRS depends on these. Examples of this kind are statistically-sound simulation extractable NIZKs [HIJ$^+$17], in which the CRS for a simulated proof is a commitment to the statement.

We formalise the notion of *game with trapdoored oracle distribution.* The concept is similar to the one in Def. 21. The difference is that now we deal with a trapdoored distribution $\mathcal{D}'$.

Definition 24 (Game with trapdoored oracle distribution). *An n-party game with trapdoored oracle distribution is a triple $\mathcal{G} := (\mathcal{D}', \mathsf{Ch})$ where*

1. *$\mathcal{D}'$ is a trapdoored distribution.*
2. *Ch is an efficient challenger: a uniform, PPT, round-based, interactive Turing machine that, for every $i \in [n]$, sends the message (Sample, i) at most once in its execution.*

Indistinguishability-preserving distributed samplers will be compatible only with a particular class of games with trapdoored oracle distribution. The interaction between the adversary and the challenger will be analogous to the one in Fig. 21 with the difference that when the challenger receives the sample R, it may also obtain the corresponding trapdoor T. The adversary instead never receives T. The choice of the auxiliary input aux' given to $\mathcal{D}'$ is made by the challenger when R is given to the adversary. We say that the game satisfies trapdoor security if it is impossible for the adversary to tell if the trapdoor was given to the challenger or not. If the first case, we say that the game is in trapdoor mode, otherwise, we say that the game is in no-trapdoor mode.

Definition 25 (Trapdoor security). *Consider an n-party game with trapdoored oracle distribution $\mathcal{G} = (\mathcal{D}', \mathsf{Ch})$. We say that $\mathcal{G}$ satisfies* trapdoor security *if every PPT adversary $\mathcal{A}$ wins the game in Fig. 23 with negligible advantage.*

Why do we need the above property? Trapdoor security ensures that, independently on whether the trapdoor will be provided, the challenger will be able to conclude its execution obtaining indistinguishable outcomes. Indistinguishability-preserving distributed samplers will guarantee that, if an game with oracle distribution $\mathcal{G}_0 = (\mathcal{D}, \mathsf{Ch}_0)$ is indistinguishable from a game with trapdoored oracle distribution $\mathcal{G}_1 = (\mathcal{D}', \mathsf{Ch}_1)$, then, also the compiled games are indistinguishable. In the security proof of our construction, we will switch the challenger of the compiled games from Ch_0 to Ch_1, using the mode of operation in which no trapdoor is given. Then, we gradually modify the output of the distributed sampler, switching from $\mathcal{D}$ to the trapdoored version $\mathcal{D}'$. In other words, there will be some hybrids in which part of the outputs of the distributed sampler are trapdoored, whereas the rest is not. Since there will be no way to predict whether the adversary chooses a trapdoored sample or not, we need to make sure that before R is delivered to it, Ch_1 will not rely on the fact that a trapdoor will be given at some point. Trapdoor security guarantees this.

Trapdoor Security Game

1. $b \xleftarrow{\$} \{0,1\}$
2. Activate the adversary $\mathcal{A}$ with $\mathbb{1}^\lambda$.
3. Receive aux and $H \subseteq [n]$ from $\mathcal{A}$.
4. Activate a new copy of Ch with $\mathbb{1}^\lambda$, H and aux.
5. Relay all the messages from Ch to $\mathcal{A}$.
6. Relay all the messages from $\mathcal{A}$ to Ch.
7. After the Ch has sent (Sample, i) for every $i \in H$, receive $\mathsf{aux}' \in \{0,1\}^{\ell(\lambda)}$ from Ch, compute $(R, T) \xleftarrow{\$} \mathcal{D}'(\mathbb{1}^\lambda, \mathsf{aux}')$ and provide $\mathcal{A}$ with R.
8. After the above occurred and after $\mathcal{A}$ has sent (Sample, i) for every $i \notin H$, provide Ch with R and, if $b = 1$, with T too.
9. Keep relaying the messages between $\mathcal{A}$ and Ch as before.

Win: The adversary wins if it guesses b.

Fig. 23. Trapdoor security game

Trapdoorable distributed samplers and compiled games. We need to explain how to compile a game with trapdoored oracle distribution. We start by introducing the concept of *trapdoorable distributed sampler.*

Definition 26 (Trapdoorable distributed sampler). *An n-party trapdoorable distributed sampler is a tuple of PPT algorithms* $(\mathsf{Setup}, \mathsf{Gen}, \mathsf{Sample}, \mathsf{SimSetup}, \mathsf{SimGen}, \mathsf{Trap})$ *where*

1. $(\mathsf{Setup}, \mathsf{Gen}, \mathsf{Sample})$ *is an n-party distributed sampler.*
2. $\mathsf{SimSetup}(\mathbb{1}^\lambda)$ *is a PPT algorithm taking as input the security parameter. The output is a simulated CRS* crs *and the information ζ.*
3. $\mathsf{SimGen}(\mathbb{1}^\lambda, \mathsf{sid}, i, \zeta, \mathsf{aux}')$ *is a PPT algorithm taking as input the security parameter, a session-identity, an index $i \in [n]$, the information ζ and* aux'*. The output is distributed sampler messages U_i and the trapdoor information ξ.*
4. $\mathsf{Trap}\big(\xi, (U_i)_{i \in [n]}\big)$ *is a deterministic algorithm taking as input the trapdoor information ξ and the distributed sampler messages $(U_i)_{i \in [n]}$. The output is a pair (R, T).*

Essentially, a trapdoorable distributed sampler is a distributed sampler in which the CRS and the messages can be simulated in a way that the outputs will be sampled from a trapdoored distribution $\mathcal{D}'$ instead of $\mathcal{D}$. In other words, the samples will be equipped with trapdoors. The latter can be retrieved from the exchanged messages using the algorithm Trap. The auxiliary information aux' needed by $\mathcal{D}'$ will be hidden in the simulated messages. All the samples produced by the construction will use the same aux'.

We can finally explain how to compile a game with trapdoored oracle distribution using a trapdoorable distributed sampler. The idea is similar to the one

Compiled Game with Trapdoored Oracle Distribution

Initialisation: This procedure is run only once, at the beginning of the game.

1. $(\mathsf{crs}, \zeta) \xleftarrow{\$} \mathsf{DS.SimSetup}(1\!\!1^\lambda)$
2. Activate the adversary $\mathcal{A}$ with $1\!\!1^\lambda$ and crs.
3. Receive a list of parties $\mathsf{ID} := \{\mathsf{id}_1, \dots, \mathsf{id}_m\}$ from $\mathcal{A}$ along with the subset of honest players $H \subseteq [m]$.

Session: This procedure can be queried multiple times and at any point of the game. Upon receiving any query $(\mathsf{NewSession}, \mathsf{tag}, \mathsf{id}_{j_1}, \dots, \mathsf{id}_{j_n}, \mathsf{aux})$ where the session identity $\mathsf{sid} := (\mathsf{tag}, \mathsf{id}_{j_1}, \dots, \mathsf{id}_{j_n})$ has not been queried before, $\mathsf{id}_{j_i} \in \mathsf{ID}$ for every $i \in [n]$, $\mathsf{id}_{j_l} \neq \mathsf{id}_{j_k}$ for every $l \neq k$ and aux, perform the following.

1. Store sid
2. $\forall i$ s.t. $j_i \in H : \quad U_i \xleftarrow{\$} \mathsf{DS.Gen}(1\!\!1^\lambda, \mathsf{sid}, i, \mathsf{crs})$
3. Activate a new copy of Ch with $1\!\!1^\lambda$, $H' := \{i \in [n] | j_i \in H\}$ and aux.
4. Relay all the messages from Ch to $\mathcal{A}$ appending sid to them.
5. Relay all the messages from $\mathcal{A}$ with prefix sid to Ch (the prefix is removed).
6. When Ch sends (Sample, i) for any $i \in H'$ except the last one left, provide $\mathcal{A}$ with $(\mathsf{sid}, \mathsf{Sample}, \mathsf{id}_{j_i}, U_i)$.
7. When Ch sends (Sample, i) for the last $i \in H'$, obtain $\mathsf{aux}' \in \{0,1\}^{\ell(\lambda)}$ from Ch_1, compute $(U_i, \xi) \xleftarrow{\$} \mathsf{DS.SimGen}(1\!\!1^\lambda, \mathsf{sid}, i, \zeta, \mathsf{aux}')$. Then, provide $\mathcal{A}$ with $(\mathsf{sid}, \mathsf{Sample}, \mathsf{id}_{j_i}, U_i)$.
8. When $\mathcal{A}$ sends $(\mathsf{sid}, \mathsf{Sample}, \mathsf{id}_{j_i}, U_i)$ for any $i \notin H'$, give (Sample, i) to Ch.
9. When all the messages $(\mathsf{sid}, \mathsf{Sample}, \mathsf{id}_{j_i}, U_i)_{i \in [n]}$ have been exchanged, compute $(R, T) \leftarrow \mathsf{DS.Trap}\big(\xi, (U_j)_{j \in [n]}\big)$. Provide (R, T) to Ch.
10. Keep relaying the messages between Ch and $\mathcal{A}$ as before.

Fig. 24. Compiled game with trapdoored oracle distribution

explained in Def. 22. The main differences is that now, the distributed sampler CRS and the last message sent by a honest party in each session are simulated using $\mathsf{SimSetup}$ and SimGen. The auxiliary information input in SimGen will be the one provided by the challenger. When all the distributed sampler messages have been exchanged, we provide the challenger with a pair (R, T) generated using Trap.

Definition 27 (Compiled game with trapdoored oracle distribution). *Consider an n-party game with trapdoored oracle distribution $\mathcal{G} = (\mathcal{D}, \mathsf{Ch})$ and let $\mathsf{DS} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Sample}, \mathsf{SimSetup}, \mathsf{SimGen}, \mathsf{Trap})$ be an n-party trapdoorable distributed sampler.*

For any PPT adversary $\mathcal{A}$, we denote by $\mathcal{A}_{\mathcal{G}'}(1\!\!1^\lambda)$ the value output by $\mathcal{A}$ at the end of the game in Fig. 24.

Defining indistinguishability-preserving distributed samplers. Indistinguishability-preserving distributed samplers compile indistinguishable games with oracle dis-

Chosen-Sample Indistinguishability for Games with Oracle Distribution

1. $b \xleftarrow{\$} \{0,1\}$
2. Activate the adversary $\mathcal{A}$ with $1\!\!1^\lambda$.
3. Receive aux and $H \subseteq [n]$ from $\mathcal{A}$.
4. Activate a copy of Ch_b with $1\!\!1^\lambda$, H and aux.
5. Relay all the messages from Ch_b to $\mathcal{A}$.
6. Relay all the messages from $\mathcal{A}$ to Ch_b.
7. After the Ch_b has sent (Sample, i) for every $i \in H$ and $\mathcal{A}$ has sent (Sample, i) for every $i \notin H$, let $\mathcal{A}$ choose R and provide it to Ch_b. Do not provide aux' to $\mathcal{A}$.
8. Keep relaying the messages between $\mathcal{A}$ and Ch_b as before.

Win: The adversary wins if it guesses b.

Fig. 25. Chosen-sample indistinguishability for games with oracle distribution

tributions into standard indistinguishable games. We are interested in the case in which one of the games with oracle distribution is trapdoored.

We define *chosen-sample indistinguishability*. Essentially, the latter says that a game with oracle distribution $\mathcal{G}_0 = (\mathcal{D}, \mathsf{Ch}_0)$ is indistinguishable from a game with trapdoored oracle distribution $\mathcal{G}_1 = (\mathcal{D}', \mathsf{Ch}_1)$ if no PPT adversary $\mathcal{A}$ can tell the two apart even if the $\mathcal{A}$ is allowed to choose the sample R. The challenger Ch_1 is never provided with trapdoors.

Definition 28 (Chosen-sample Indistinguishable games with oracle distribution). *Consider any pair $(\mathcal{G}_0, \mathcal{G}_1)$ where $\mathcal{G}_0 = (\mathcal{D}, \mathsf{Ch}_0)$ is a game with oracle distribution and $\mathcal{G}_1 = (\mathcal{D}', \mathsf{Ch}_1)$ is a game with trapdoored oracle distribution. We say that $\mathcal{G}_0$ and $\mathcal{G}_1$ are chosen-sample indistinguishable if every PPT adversary $\mathcal{A}$ wins the game in Fig. 25 with negligible advantage.*

The reason why we let the adversary choose R is the influence allowed in the compiled games. While in a game with oracle distribution the choice of the sample R is not affected by the adversary, in the compiled games, the adversary has always some influence. If we want the compiled games to be indistinguishable, it is important that the challengers Ch_0 and Ch_1 cannot be told apart, no matter how the adversary influences the choice of R.

We can finally define indistinguishability-preserving distributed samplers.

Definition 29 (Indistinguishability-preserving distributed sampler). *Let $\mathcal{D}(1\!\!1^\lambda)$ be an efficient distribution and let $\mathcal{D}'$ be a trapdoored distribution for $\mathcal{D}$. We say that an n-party trapdoorable distributed sampler is indistinguishability-preserving for $(\mathcal{D}, \mathcal{D}')$ against* AClass *if, for every PPT adversary $\mathcal{A} \in$* AClass *and for every pair $(\mathcal{G}_0, \mathcal{G}_1)$ of chosen-sample indistinguishable games where $\mathcal{G}_0 = (\mathcal{D}, \mathsf{Ch}_0)$ is a game with oracle distribution and $\mathcal{G}_1 = (\mathcal{D}', \mathsf{Ch}_1)$ is a game with* *trapdoored oracle distribution satisfying trapdoor security, we have*

$$\left|\Pr[\mathcal{A}_{\mathcal{G}'_0}(\mathbb{1}^\lambda) = 1] - \Pr[\mathcal{A}_{\mathcal{G}'_1}(\mathbb{1}^\lambda) = 1]\right| = \mathsf{negl}(\lambda),$$

where $\mathcal{G}'_0$ and $\mathcal{G}'_1$ are the compiled games.

Applications of indistinguishability-preserving distributed samplers for protocol security. We now show that, in most cases, indistinguishability-preserving distributed samplers can be used to remove CRSs in MPC protocols at the cost of one additional round of interaction while preserving simulation security. This holds in a context of active adversaries statically corrupting any number of the parties. Our theorem is formalised below.

Theorem 14. *Assume the existence of authenticated point-to-point channels and a broadcast medium. Let Π be an n-party protocol implementing a PPT functionality $\mathcal{F}$ against active PPT adversaries in the UC model with static corruption. Suppose that Π relies on a CRS R generated according to the distribution $\mathcal{D}(\mathbb{1}^\lambda)$. Let $\mathcal{S}$ be the corresponding PPT simulator.*

Suppose that $\mathcal{S}$ can be regarded as the sequential composition of $\mathcal{S}_1$ and $\mathcal{S}_2$ where $\mathcal{S}_1$ never interacts with the functionality, generates a pair $(R, T) \xleftarrow{\$} \mathcal{D}'(\mathbb{1}^\lambda)$ and provides the adversary with the simulated CRS R and $\mathcal{S}_2$ with (R, T).

Assume that $\mathcal{D}'$ is a trapdoored distribution for $\mathcal{D}$. Let DS be an n-party indistinguishability-preserving distributed sampler for $(\mathcal{D}, \mathcal{D}')$. Let Π' be the sequential composition of DS with Π. Then, Π' implements $\mathcal{F}$ against active PPT adversaries in the UC model with static corruption.

Observe that the round complexity of the protocol Π' has only increased by one. The idea at the base of the proof is rather immediate: the protocol Π can be reformulated as a game with oracle distribution $\mathcal{G}_0$. In the latter, the special messages are all exchanged at the beginning of the session. In a similar way, the simulation can be reformulated as a game with trapdoored oracle distribution $\mathcal{G}_1$ in which the auxiliary information given to $\mathcal{D}'$ is the empty string. To be precise, the simulation of Π corresponds to the trapdoor mode of $\mathcal{G}_1$, the no-trapdoor mode of $\mathcal{G}_1$ is instead identical to $\mathcal{G}_0$. Trapdoor security is an immediate consequence of the UC-security of Π. Chosen-sample indistinguishability is instead for free as $\mathcal{G}_0$ and the no-trapdoor mode of $\mathcal{G}_1$ are identical. That is enough to argue that the compiled games $\mathcal{G}'_0$ and $\mathcal{G}'_1$ are indistinguishable too. It is straightforward to notice that if we reformulate the compiled protocol Π' as a game, we obtain $\mathcal{G}'_0$. To terminate the proof, we notice that $\mathcal{G}'_1$ easily leads to a simulator $\mathcal{S}'$ for Π' and $\mathcal{F}$.

Proof. Let H be the set of honest parties. For every $i \in [n]$, let id_i denote the identity of the i-th party.

A single real-world execution of Π can be formulated as a n-party game with oracle distribution $\mathcal{G}_0$. In such game, the challenger Ch_0 immediately sends (Sample, i) for every $i \in H$. Then, it waits for the adversary to send (Sample, i)

for every $i \notin H$. It ignores all other communications received before that. Then, Ch_0 runs the protocol Π with $\mathcal{A}$ on behalf of the honest parties.

In a similar way, a single ideal-world execution, can be rephrased as a n-party game with trapdoor oracle distribution $\mathcal{G}_1 = (\mathcal{D}', \mathsf{Ch}_1)$ where the challenger Ch_1 behaves as follows:

1. It immediately sends (Sample, i) for every $i \in H$, it sets aux' to be the empty string.
2. It waits for the adversary to send (Sample, i) for every $i \notin H$. It ignores all other communications received before that.
3. If it receives only a sample R, it executes Ch_0 providing it with R
4. It it receives a pair (R, T), it runs $\mathcal{S}_2$ along with $\mathcal{F}$.

By the UC security of Π, $\mathcal{G}_1$ satisfies trapdoor security. Moreover, it is immediate to see that the games $\mathcal{G}_0$ and $\mathcal{G}_1$ are perfectly chosen-sample indistinguishable.

Since DS is indistinguishability-preserving, the compiled games $\mathcal{G}_0'$ and $\mathcal{G}_1'$ are still indistinguishable. Observe that if we reformulate the real-world execution of Π', we obtain $\mathcal{G}_0'$.

We now consider the simulator $\mathcal{S}'$ that generates the distributed sampler CRS crs using $(\mathsf{crs}, \zeta) \xleftarrow{\$} \mathsf{SimSetup}(\mathbb{1}^\lambda)$. In every session $\mathsf{sid} = (\mathsf{tag}, \mathsf{id}_{j_1}, \dots, \mathsf{id}_{j_n})$ of the protocol Π' where $\mathsf{id}_{j_1}, \dots, \mathsf{id}_{j_n}$ denote the identities of the parties involved, $\mathcal{S}'$ performs the following operations

1. pick i such that $j_i \in H$
2. $\forall l \neq i$ s.t. $j_l \in H : \quad U_l \xleftarrow{\$} \mathsf{DS.Gen}(\mathbb{1}^\lambda, \mathsf{sid}, l, \mathsf{crs})$
3. $(U_i, \xi) \xleftarrow{\$} \mathsf{SimGen}(\mathbb{1}^\lambda, \mathsf{sid}, i, \zeta)$
4. send $(U_l)_{j_l \in H}$ to the adversary on behalf of the honest parties
5. wait for $(U_l)_{j_l \notin H}$ from the adversary
6. $(R, T) \leftarrow \mathsf{Trap}\big(\xi, (U_l)_{l \in [n]}\big)$
7. run $\mathcal{S}_2(\mathbb{1}^\lambda, R, T)$ interacting with the functionality $\mathcal{F}$ and the adversary.

Observe that if we reformulate the interaction between $\mathcal{F}$, $\mathcal{S}'$ and the adversary as a game, we obtain $\mathcal{G}_1'$. We conclude that no active PPT adversary can distinguish between Π' and the composition of $\mathcal{F}$ and $\mathcal{S}'$. This terminates the proof. □

In some cases, when the first round of interaction in Π is independent of the CRS, indistinguishability-preserving distributed samplers allow removing the CRS without affecting the round complexity. The result is formalised below.

Theorem 15. *Assume the existence of authenticated point-to-point channels and a broadcast medium. Let Π be an n-party protocol implementing a PPT functionality $\mathcal{F}$ against active PPT adversaries in the UC model with static corruption. Let $\mathcal{S}$ be the corresponding PPT simulator. Suppose that Π can be rewritten as the sequential composition of a one-round protocol Π_1 with no CRS and a protocol Π_2 that relies on a CRS R generated according to the distribution $\mathcal{D}(\mathbb{1}^\lambda)$.*

Suppose that $\mathcal{S}$ can be regarded as the sequential composition of $\mathcal{S}_1$, $\mathcal{S}_2$ and $\mathcal{S}_3$ where:

- $\mathcal{S}_1$ *never interacts with the functionality, generates values* $(R, T) \xleftarrow{\$} \mathcal{D}'(\mathbb{1}^\lambda)$ *and provides the adversary with the simulated CRS* R *and* $\mathcal{S}_3$ *with* (R, T).
- $\mathcal{S}_2$, *never interacts with the functionality, generates the first-round messages of the honest parties using* Π_1 *and delivers them to the adversary. It passes its internal state to* $\mathcal{S}_3$.

Assume that $\mathcal{D}'$ *is a trapdoored distribution for* $\mathcal{D}$. *Let* DS *be an* n*-party indistinguishability-preserving distributed sampler for* $(\mathcal{D}, \mathcal{D}')$. *Let* Π' *be the composition of* DS *with* Π *where* DS *and* Π_1 *are run in parallel. Then,* Π' *implements* $\mathcal{F}$ *against active PPT adversaries in the UC model with static corruption.*

The proof of Theorem 15 follows the blueprint of the proof of Theorem 14. Once again, we reformulate Π as a game with oracle distribution $\mathcal{G}_0$. This time the special messages are all sent simultaneously with the first round of communications. Since the simulator $\mathcal{S}$ generates the first round messages exactly as in Π, we can design a game with trapdoored oracle distribution $\mathcal{G}_1$ in which the trapdoor mode is a reformulation of the ideal world whereas the no-trapdoor mode is identical to $\mathcal{G}_0$. Trapdoor security is a consequence of the UC-security of Π, chosen-sample indistinguishability instead comes for free as before. The rest remains as in the proof of Theorem 14.

Proof. Let H be the set of honest parties. For every $i \in [n]$, let id_i denote the identity of the i-th party.

As before, a single real-world execution of Π can be formulated as a n-party game with oracle distribution $\mathcal{G}_0$. In such game, the challenger Ch_0 immediately sends (Sample, i) for every $i \in H$. Simultaneously, it sends the messages of the honest parties in protocol Π_1. Then, it waits for the adversary to send (Sample, i) for every $i \notin H$ along with the messages of the corrupted players in Π_1. Finally, Ch_0 runs the protocol Π_2 with $\mathcal{A}$ on behalf of the honest parties.

In a similar way, a single ideal-world execution, can be rephrased as a n-party game with trapdoor oracle distribution $\mathcal{G}_1 = (\mathcal{D}', \mathsf{Ch}_1)$ where the challenger Ch_1 behaves as follows:

1. It runs $\mathcal{S}_2$. The messages generated by $\mathcal{S}_2$ are delivered to the adversary in conjunction with (Sample, i) for every $i \in H$. The challenger Ch_1 also outputs the empty string aux'.
2. It waits for the adversary to send (Sample, i) for every $i \notin H$, along with the first-round messages of the corrupted parties.
3. If it receives only a sample R, it executes Π_2 on behalf of the honest parties using R as CRS.
4. It it receives a pair (R, T), it runs $\mathcal{S}_3$ along with $\mathcal{F}$. The simulator $\mathcal{S}_3$ is given (R, T) and the messages of the corrupted players in Π_1.

Notice that if Ch_1 receives R but not the trapdoor T, the view of the adversary is the same as in Π. So, by the UC security of Π, $\mathcal{G}_1$ satisfies trapdoor security. Moreover, it is immediate to see that the games $\mathcal{G}_0$ and $\mathcal{G}_1$ are perfectly chosen-sample indistinguishable.

Since DS is indistinguishability preserving, the compiled games $\mathcal{G}_0'$ and $\mathcal{G}_1'$ are still indistinguishable. Observe that if we reformulate the real-world execution of Π' as a game, we obtain $\mathcal{G}_0'$.

We now consider the simulator $\mathcal{S}'$ that generates the distributed sampler CRS crs using $(\mathsf{crs}, \zeta) \xleftarrow{\$} \mathsf{SimSetup}(\mathbb{1}^\lambda)$. In every session $\mathsf{sid} = (\mathsf{tag}, \mathsf{id}_{j_1}, \dots, \mathsf{id}_{j_n})$ of the protocol Π' where $\mathsf{id}_{j_1}, \dots, \mathsf{id}_{j_n}$ denote the identities of the parties involved, $\mathcal{S}'$ performs the following operations

1. pick i such that $j_i \in H$
2. $\forall l \neq i$ s.t. $j_l \in H: \quad U_l \xleftarrow{\$} \mathsf{DS.Gen}(\mathbb{1}^\lambda, \mathsf{sid}, l, \mathsf{crs})$
3. $(U_i, \xi) \xleftarrow{\$} \mathsf{SimGen}(\mathbb{1}^\lambda, \mathsf{sid}, i, \zeta)$
4. generate the first-round messages of the honest parties in Π_1 following the protocol. Provide $\mathcal{S}_3$ with the view of the honest players.
5. send $(U_l)_{j_l \in H}$ to the adversary along with the messages generated in the previous step.
6. wait for $(U_l)_{j_l \notin H}$ and the corrupted player messages in Π_1 from the adversary
7. $(R, T) \leftarrow \mathsf{Trap}\big(\xi, (U_l)_{l \in [n]}\big)$
8. run $\mathcal{S}_3(\mathbb{1}^\lambda, R, T)$ interacting with the functionality $\mathcal{F}$ and the adversary. $\mathcal{S}_3$ is also given the messages of the corrupted players in Π_1.

Observe that if we reformulate the interaction between $\mathcal{F}$, $\mathcal{S}'$ and the adversary as a game, we obtain $\mathcal{G}_1'$. We conclude that no active PPT adversary can distinguish between Π' and the composition of $\mathcal{F}$ and $\mathcal{S}'$. This terminates the proof. □

Generalisations. Sometimes, indistinguishability-preserving distributed samplers can be used to remove CRSs even from UC-secure protocols that satisfy neither of the hypothesis of Theorem 14 and Theorem 15. For instance, in some cases, we can let the simulated CRS depend on auxiliary information aux' provided by the functionality. In order for the proofs to go through, however, we need to ask that indistinguishability between real world and ideal world holds even when aux' is leaked to the adversary.

Theorem 15 can also be generalised in the sense that the simulator $\mathcal{S}$ does not strictly need to follow the protocol in the first round. The important thing, indeed, is to be able to successfully terminate the simulation even if $\mathcal{S}_1$ abruptly refuses to provide the trapdoor T and instead provides a sample R chosen by the adversary ($\mathcal{S}$ can even ask the functionality $\mathcal{F}$ to reveal its internal state when that happens). That would ensure chosen-sample indistinguishability.

The limits of indistinguishability-preserving. Although indistinguishability-preserving distributed samplers allow removing CRSs from a broad range of UC secure protocols, we know that there exist constructions for which this fails. One example in the protocol $\Pi_{\mathcal{D}}$ in which, after being provided with a CRS R sampled according to $\mathcal{D}(\mathbb{1}^\lambda)$, all the parties output R. This protocol trivially implements the functionality $\mathcal{F}_{\mathcal{D}}$ that generates a sample from $\mathcal{D}(\mathbb{1}^\lambda)$ and provides it to all the parties. If indistinguishability-preserving distributed samplers worked for this case we would obtain a distributed sampler for $\mathcal{D}$ satisfying the simulation-based definition of [ASY22]. We know that this is impossible [ASY22,AOS23].

6 Lossy Distributed Samplers

In this section, we introduce a new variant of distributed sampler called *lossy distributed samplers*. On their own, lossy distributed samplers are not sufficient to achieve hardness or indistinguishability preservation. However, they are a useful stepping stone towards our goal.

The construction of [ASY22] and its problems with rushing adversaries. In [ASY22], Abram, Scholl and Yakoubov presented a distributed sampler achieving security against semi-malicious non-rushing adversaries in the UC model. In other words, the protocol implements the ideal functionality that provides all the parties with a random sample from $\mathcal{D}(1\!\!1^\lambda)$. The construction does not rely on random oracles nor CRSs.

There is a property that allows all this: output programming. Specifically, given any distributed sampler messages $(\hat{U}_j)_{j\notin H}$ for the corrupted parties and a random sample $\hat{R}$ from $\mathcal{D}(1\!\!1^\lambda)$, it is possible to generate fake messages for the honest parties such that, when used in conjunction with $(\hat{U}_j)_{j\notin H}$, the output of the protocol is $\hat{R}$. These fake messages are indistinguishable from the real ones, so no adversary is able to tell if the output was programmed or not.

This property is sufficient to achieve security against non-rushing adversaries in the UC model. Indeed, in this setting, the simulator gets to know the messages of the corrupted parties before generating those of the honest players. So, it can just send fake messages that are programmed to output $\hat{R}$, the sample received from the functionality. In some sense, the simulator is leveraging rushing against the adversary.

The strategy, however, fails against rushing adversaries. Now, indeed, the adversary receives the honest messages first and then it chooses what to send on behalf of the corrupted players. The simulator can still try to program some of the outputs of the distributed sampler, but it can apply the technique only a limited number of times: the samples provided by the functionality have large entropy and so, it is impossible to "hide" many of them in the messages of the honest parties. In conclusion, if the messages of the corrupted players are chosen at random, the simulator cannot predict the choice of the adversary, so, with overwhelming probability, the output of the protocol will not have been programmed.

Unfortunately, the issue we highlighted is not only restricted to the construction of [ASY22], it is part of a more general problem formalised by Abram, Obremski and Scholl in [AOS23]: without random oracle, distributed samplers with UC security against rushing adversaries are essentially impossible. The reason is that, in the protocol, we would like the entropy of the output conditioned on the messages of any subset S of the parties to be high, i.e. $\mathsf{H}\big(R|(U_i)_{i\in S}\big) = \omega(\log\lambda)$. If that was not the case, an adversary corrupting all the parties in S would have too much influence over the output of the protocol, compromising security. On the other hand, in the ideal world, we would like the simulator to generate fake honest messages so that $\mathsf{H}\big(R|(U_i)_{i\in H}\big)$ is small, namely $O(\log\lambda)$. In this way, we can hope to hide ideal samples in the output space so that, even if the adversary decides the messages of the corrupted parties after seeing $(U_i)_{i\in H}$, the output of the protocol will be an ideal sample with high probability. The results presented by Abram, Obremski and Scholl in [AOS23] suggest that, for any such simulator, it is possible to distinguish between the real $(U_i)_{i\in H}$ and the simulated ones.

Introducing lossy distributed samplers. We move back to our goal: building hardness-preserving and indistinguishability-preserving distributed samplers. Although we are not aiming for UC security anymore, having a way to control the output of the distributed sampler is still a desirable property that would simplify our task. In this context, the discussion about entropy in the previous paragraph raised a point we need to face. We do this by introducing the notion of *lossy distributed sampler*.

A lossy distributed sampler is a distributed sampler having two modes of operation. In the standard mode, for every non-empty $H \subseteq [n]$, the entropy $\mathsf{H}\big(R|(U_i)_{i\in H}\big)$ will remain high, namely $\omega(\log\lambda)$. In this way, we can make sure that the influence of the adversary on the protocol is limited. By switching to lossy mode, however, the messages of the honest parties restrict the output in a set of polynomial size, with high probability. In other words, in the lossy mode, the outputs of the protocol becomes predictable. This allows us to deal with rushing.

Unavoidably, an adversary can always distinguish between a distributed sampler in standard mode and one in lossy mode [11]. However, lossy distributed samplers permit making the distinguishability advantage arbitrarily small: for every polynomial $p(\lambda)$ and inverse polynomial function $\delta(\lambda)$, we can set the parameters of the lossy mode so that no adversary running in time at most $p(\lambda)$ can distinguish between the standard mode and the lossy mode with advantage greater than $\delta(\lambda)$. Observe that this property strongly resembles the one of ELFs [Zha16]. This is not a coincidence, as ELFs will be one of the building blocks for lossy distributed samplers.

We now present the precise definition.

Definition 30 (Lossy distributed sampler). *An* lossy distributed sampler *for* AClass *is an n-party distributed sampler* $\mathsf{DS} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Sample})$ *for which there exists a tuple of PPT algorithms* $(\mathsf{LossySetup}, \mathsf{LossyGen}, \mathsf{Project}, \mathsf{Extract})$ *with the following syntax:*

- $\mathsf{LossySetup}$ *is randomised and takes as input the security parameter and an integer $q \in \mathbb{N}$. The output is a lossy distributed sampler* crs*, the CRS trapdoor* ζ*.*
- $\mathsf{LossyGen}$ *is uniform, randomised and takes as input the security parameter, a session identity* sid*, an index $i \in [n]$ and the CRS information ζ. The output is a lossy distributed sampler message U_i and the extraction trapdoor* ξ*.*

[11] In the standard mode, running the protocol twice produces different outputs with overwhelming probability, in lossy mode, instead, there is a non-negligible probability of obtaining a collision.

- $\mathsf{Project}$ *is uniform, deterministic and takes as input the CRS trapdoor* ζ*,* n *distributed sampler messages* $(U_i)_{i\in[n]}$ *and a session identity* sid*. The output is an element* z*.*
- $\mathsf{Extract}$ *is uniform, deterministic and takes as input an extraction trapdoor* ξ *and a value* z*. The output is a sample* R*.*

A lossy distributed sampler satisfies the following properties.

- ***(Arbitrarily small advantage).*** *For every polynomial* $p(\lambda)$ *and inverse polynomial function* $\delta(\lambda)$*, there exists a polynomial* $q(\lambda)$ *such that every adversary* $\mathcal{A} \in \mathsf{AClass}$ *running in time at most* p *can win the game in Fig. 26 with advantage asymptotically smaller than* δ*.*
- ***(Small support).*** *For every polynomial* $q(\lambda)$*, there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that, for every session identity* sid *and index* $i \in [n]$*,*

$$\Pr\left[\big|\mathsf{Supp}_\zeta\big| > q(\lambda) \middle| \begin{matrix}(\mathsf{crs}, \zeta) \xleftarrow{\$} \mathsf{LossySetup}\big(1\!\!1^\lambda, q(\lambda)\big) \\ (U_i, \xi) \xleftarrow{\$} \mathsf{LossyGen}(1\!\!1^\lambda, \mathsf{sid}, i, \zeta)\end{matrix}\right] \le \mathsf{negl}(\lambda),$$

where $\mathsf{Supp}_\zeta := \big\{\mathsf{Project}\big(\zeta, (U_j)_{j\in[n]}, \mathsf{sid}\big) \big| \big(\mathsf{sid}, (U_j)_{j\in[n]}\big) \in \{0,1\}^*\big\}$*.*

Notice that the lossy mode is split into two parts: a lossy setup $\mathsf{LossySetup}$ and a lossy generation algorithm $\mathsf{LossyGen}$. The lossy setup takes as input the parameter $q(\lambda)$ and outputs a fake CRS along with a trapdoor ζ. The lossy generation algorithm takes as input the trapdoor ζ and the index of party $i \in [n]$. The output is the lossy message for P_i. In order to switch the distributed sampler to lossy mode, it is sufficient that a single party sends a message in lossy mode. When that happens, with high probability, the output of Sample is obtained by first projecting the exchanged messages in a set of polynomial size and then deterministically mapping the result into a sample from $\mathcal{D}(1\!\!1^\lambda)$. Observe, however, that our definition does not guarantee that this occurs with overwhelming probability, but just with probability $1 - \delta(\lambda)$, where $\delta(\lambda)$ is an arbitrarily small inverse-polynomial quantity. Informally, this means that the lossy mode restricts most of the outputs of the construction in a set of polynomial size.

In order to make the distinguishability advantage between standard and lossy mode arbitrarily small, it is important that $\mathsf{Project}$ and $\mathsf{Extract}$ are hard to compute when the trapdoors ζ and ξ are kept secret.

Regularity of lossy distributed samplers. We now formulate the definition of regular lossy distributed sampler. Essentially, this consists of a lossy distributed sampler for which the output of the lossy mode is predictable with inverse-polynomial probability independently of the behaviour of the adversary. Observe that if the output space was not restricted in a set of polynomial size, this property was unachievable.

Definition 31 (Regularity). *We say that a lossy distributed sampler* (Setup, Gen, Sample, $\mathsf{LossySetup}$, $\mathsf{LossyGen}$, $\mathsf{Project}$, $\mathsf{Extract}$) *is regular if there exists a uni-*

Lossy distributed sampler game

Initialisation: This procedure is run only once, at the beginning of the game.

1. $b \xleftarrow{\$} \{0,1\}$
2. $\mathsf{crs}_0 \xleftarrow{\$} \mathsf{Setup}(\mathbb{1}^\lambda)$
3. $(\mathsf{crs}_1, \zeta) \xleftarrow{\$} \mathsf{LossySetup}\big(\mathbb{1}^\lambda, q(\lambda)\big)$
4. Activate $\mathcal{A}$ with $\mathbb{1}^\lambda$ and crs_b
5. Receive a set of distinct identities $\mathsf{ID} := (\mathsf{id}_i)_{i\in[m]}$ from $\mathcal{A}$.

New session: This procedure can be queried multiple times and at any point of the game. Upon receiving any query $(\mathsf{NewSession}, \mathsf{tag}, \mathsf{id}_{j_1}, \ldots, \mathsf{id}_{j_n}, i)$ where the session identity $\mathsf{sid} := (\mathsf{tag}, \mathsf{id}_{j_1}, \ldots, \mathsf{id}_{j_n})$ has never been queried before, $\mathsf{id}_{j_l} \in \mathsf{ID}$ for every $l \in [n]$ and all $\mathsf{id}_{j_l} \neq \mathsf{id}_{j_k}$ for every $l \neq k$, compute the following.

1. $U_i^0 \xleftarrow{\$} \mathsf{Gen}\big(\mathbb{1}^\lambda, \mathsf{sid}, i, \mathsf{crs}\big)$
2. $(U_i^1, \xi) \xleftarrow{\$} \mathsf{LossyGen}\big(\mathbb{1}^\lambda, \mathsf{sid}, i, \zeta\big)$
3. Provide the adversary with $U_i := U_i^b$
4. Store $(\mathsf{sid}, i, U_i, \xi)$

Sample: This procedure can be queried multiple times and at any point of the game. Upon receiving any query $\big(\mathsf{Sample}, \mathsf{sid}, (U_j)_{j\neq i}\big)$ where sid denotes the identity of an already initiated session, compute the following. The same session identity can be queried multiple times.

1. Retrieve (sid, i, U_i)
2. $R_0 \leftarrow \mathsf{Sample}(U_1, \ldots, U_n, \mathsf{sid}, \mathsf{crs}_0)$
3. $R_1 \leftarrow \mathsf{Extract}\Big(\xi, \mathsf{Project}\big(\zeta, (U_j)_{j\in[n]}, \mathsf{sid}\big)\Big)$
4. Provide the adversary with R_b.

Win: The adversary wins if it guesses b

Fig. 26. Lossy distributed sampler game

form PPT algorithm $\mathcal{Z}$ and a polynomial $s(\lambda, q)$ such that, for every polynomial $q(\lambda)$, with overwhelming probability over the randomness of $(\mathsf{crs}, \zeta) \xleftarrow{\$} \mathsf{LossySetup}\big(\mathbb{1}^\lambda, q(\lambda)\big)$,

$$\Pr_{\mathcal{Z}}\big[\mathcal{Z}(\zeta) = \mathsf{Project}\big(\zeta, (U_j)_{j\in[n]}, \mathsf{sid}\big)\big] \geq \frac{1}{s\big(\lambda, q(\lambda)\big)}$$

for every $\big(\mathsf{sid}, (U_i)_{i\in[n]}\big) \in \{0,1\}^*$, where the above probability is taken only over the randomness of $\mathcal{Z}$.

Formally, the above definition states that $\mathcal{Z}$ allows to predict the output of the projection with inverse-polynomial probability. Furthermore, the success probability is essentially only over the randomness of $\mathcal{Z}$. That immediately allows predicting the output thanks to $\mathsf{Extract}$.

Programmability. We finally formalise the notion of *programmable lossy distributed sampler*. This consists of a construction in which the lossy mode allows hiding an ideal sample R in the output space. In particular, there will be an

Programmability Game

Each phase is run only once.

Initialisation Phase:

1. $b \xleftarrow{\$} \{0,1\}$
2. $(\mathsf{crs}, \zeta) \xleftarrow{\$} \mathsf{LossySetup}\big(\mathbb{1}^\lambda, q(\lambda)\big)$
3. Activate $\mathcal{A}$ with $\mathbb{1}^\lambda$, crs and ζ.

Generation Phase:

1. Receive $i \in [n]$, sid and z from the adversary.
2. $R \xleftarrow{\$} \mathcal{D}(\mathbb{1}^\lambda)$
3. $(U_i^0, \xi^0) \xleftarrow{\$} \mathsf{LossyGen}\big(\mathbb{1}^\lambda, \mathsf{sid}, i, \zeta\big)$
4. $(U_i^0, \xi^1) \xleftarrow{\$} \mathsf{ProgGen}\big(\mathbb{1}^\lambda, \mathsf{sid}, i, z, R, \zeta\big)$
5. Provide the adversary with $U_i := U_i^b$

Sampling Phase:

1. Receive $(U_j)_{j \neq i}$ from the adversary
2. $R_0 \leftarrow \mathsf{Extract}\Big(\xi^0, \mathsf{Project}\big(\zeta, (U_j)_{j\in[n]}, \mathsf{sid}\big)\Big)$
3. $R_1 \leftarrow \mathsf{Extract}\Big(\xi^1, \mathsf{Project}\big(\zeta, (U_j)_{j\in[n]}, \mathsf{sid}\big)\Big)$
4. If $\mathsf{Project}\big(\zeta, (U_j)_{j\in[n]}, \mathsf{sid}\big) = z$ and $z \neq \perp$, set $R_1 \leftarrow R$.
5. If $\mathsf{Project}\big(\zeta, (U_j)_{j\in[n]}, \mathsf{sid}\big) = \perp$, set $R_1 \leftarrow \perp$.
6. Provide the adversary with R_b.

Win: The adversary wins if it guesses b

Fig. 27. Programmability game

element z such that executions that are projected to z will output R with high probability. Furthermore, the adversary will not be able to tell if one of the outputs was programmed even if we provide it with z and the trapdoor ζ. The extraction trapdoor ξ will instead remain secret.

Observe that programmability is the only property that guarantees that the outputs of the distributed sampler look like those of the targetted distribution $\mathcal{D}(\mathbb{1}^\lambda)$ and no further information is leaked. In Section 8, we will show that lossy distributed samplers that are regular and programmable are hardness-preserving.

Definition 32 (Programmability). *We say that a lossy distributed sampler* $(\mathsf{Setup}, \mathsf{Gen}, \mathsf{Sample}, \mathsf{LossySetup}, \mathsf{LossyGen}, \mathsf{Project}, \mathsf{Extract})$ *for* $\mathcal{D}$ *is programmable if there exists a uniform PPT algorithm* $\mathsf{ProgGen}$ *such that no PPT adversary in* AClass *can win the game in Fig. 27 with non-negligible advantage.*

7 Building Lossy Distributed Samplers

In this section, we present a lossy distributed sampler that is regular and programmable. In the non-uniform setting, the construction relies on a uniformly random CRS which can be reused multiple times. In the uniform setting, instead, we need no CRS. Security is based, among other primitives, on subexponentially secure indistinguishability obfuscation and multi-key FHE. We achieve security against any active adversary statically corrupting up to $n-1$ parties.

The construction of [ASY22]. Our starting point is the semi-malicious distributed sampler of [ASY22, Section 4], which achieves security against non-rushing adversaries in the plain model.

Our construction inherits the same structure: the distributed sampler message of each party P_i consists of two obfuscated programs. The purpose of the first one is to generate a pseudorandom string s_i and encrypt it under a multi-key FHE public key pk_i. The random string s_i will be P_i's share of the randomness input into $\mathcal{D}(\mathbb{1}^\lambda)$. In other words, the output of the distributed sampler will be a sample R obtained by adding the strings $s_1, s_2, \dots, s_n$ and feeding the result as randomness for $\mathcal{D}(\mathbb{1}^\lambda)$. We call this first program *the encryption program* of party P_i and we denote it by EP_i.

The second program instead has the purpose of applying homomorphic operations on the ciphertexts generated by the encryption programs, deriving an encryption C of the output R. The program terminates its execution outputting a partial decryption of C using the private counterpart of pk_i. We call this second program *the decryption program* of party P_i and we denote it by DP_i. The encryption of s_j and the public key pk_j will be derived running EP_j inside the code of DP_i. The encryption program EP_j will be given as input to DP_i for every $j \neq i$.

To summarise, in order to obtain a random sample R, the parties just feed each decryption program DP_i with the encryption programs $(\mathsf{EP}_j)_{j\neq i}$. In this way, they obtain the partial plaintext d_i. The output will be derived by performing the final decryption $R \leftarrow \mathsf{FinDec}(d_1, \dots, d_n)$.

Counteracting the residual function attack. A common issue of one-round MPC protocols is residual function attacks: the adversary can rerun the protocol in its head keeping the same messages for the honest parties but using different messages for the corrupted players. In this way, it obtains a different output that might be correlated to the original one. Observe that the adversary can repeat this attack as many times as it likes, potentially obtaining a lot of leakage.

In order to prevent this issue in their distributed sampler [ASY22], Abram, Scholl and Yakoubov made sure that EP_i encrypts an independent-looking s_i for every choice of $(\mathsf{EP}_j)_{j\neq i}$. They achieved this by letting every party P_i choose a hash key hk_i and providing EP_i with a digest y_i of $(\mathsf{hk}_j, \mathsf{EP}_j)_{j\neq i}$ under hk_i (notice that we cannot directly input $(\mathsf{EP}_j)_{j\neq i}$ into EP_i as the former is significantly larger than the latter). The encryption program EP_i will derive s_i by feeding y_i into a puncturable PRF. The key used for the encryption will also change depending on $(\mathsf{EP}_j)_{j \neq i}$. The technique remains the same as before: by feeding y_i into another puncturable PRF, the program obtains randomness r_i and r'_i that will be used for the key generation and the encryption. The hash keys will be broadcast by the parties as part of their message.

Using this strategy, even if an adversary reruns the distributed sampler protocol in its head changing any $(\mathsf{hk}_j, \mathsf{EP}_j)$, the encryption program EP_i will generate an independent looking s_i and so the new output R' obtained by the adversary will look independent of the original one. Notice that changing any DP_j instead does not help in learning information about R.

The construction in this paper will keep using the technique of [ASY22]. We sketch the unobfuscated code of the encryption program EProg in Fig. 28.

```
$\mathsf{EProg}[K_1^{(i)}, K_2^{(i)}, i]$

**Hard-coded.** The PPRF keys $K_1^{(i)}$ and $K_2^{(i)}$, the index $i$.
**Input.** A digest $y \in \{0,1\}^{t(\lambda)}$.

1. $s_i \leftarrow F_1(K_1^{(i)}, y)$
2. $(r_i, r'_i, r''_i, \eta_i, \eta'_i) \leftarrow F_2(K_2^{(i)}, y)$
3. $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{mkFHE.Gen}(1\!\!1^\lambda, i;\, r_i)$
4. $c_i \leftarrow \mathsf{mkFHE.Enc}(\mathsf{pk}_i, s_i;\, r'_i)$
5. Output $(\mathsf{pk}_i, c_i)$.
```

Fig. 28. The unobfuscated encryption program of party P_i

Adjustments in the decryption programs. The modifications to the encryptions programs we added in the previous paragraph require minor adjustments in the decryption programs. As we have mentioned, for every $j \in [n]$, each DP_i needs to evaluate the encryption program EP_j to obtain pk_j and the encryption of s_j. In order to do this, it needs to compute the digest y_j that will be fed into EP_j. For this reason, we need to provide DP_i not only with $(\mathsf{EP}_j)_{j \neq i}$ but also with all the hash keys $(\mathsf{hk}_j)_{j \neq i}$. The pair $(\mathsf{hk}_i, \mathsf{EP}_i)$ will instead be hardcoded into DP_i. In the decryption program, we also hardcode the PRF key that produced the randomness for the key generation in EP_i. This will allow DP_i to retrieve the secret key needed for the partial decryption.

We also introduce another modification to the decryption programs and the construction in general. The reason for this will be clearer after reading the next paragraphs. Along with hk_i, EP_i and DP_i, each party P_i will now broadcast an almost everywhere extractable NIZK π_i proving the well-formedness of $(\mathsf{hk}_i, \mathsf{EP}_i)$. In the non-uniform case, this NIZK will require a CRS. Luckily, the latter can be uniformly random (see Section 4.1 and Section 10.1). We denote the construction by NIZK. Each decryption program DP_i will now receive the

$\mathsf{DProg}[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma]$

Hard-coded. The index i of the party, the session identity sid, a PPRF key $K_2^{(i)}$, the encryption program EP_i, the hash key hk_i, the CRS for the extractable NIZK σ.
Input. Set of $n-1$ tuples $(\mathsf{hk}_j, \mathsf{EP}_j, \pi_j)_{j \neq i}$.

```
$\forall j \neq i : \quad b_j \leftarrow \mathsf{NIZK.Verify}\big(\sigma, (\mathsf{sid}, j), \pi_j, (j, \mathsf{hk}_j, \mathsf{EP}_j)\big)$
If $\exists j \neq i$ such that $b_j = 0$, output $\perp$
$\forall j \in [n] : \quad y_j \leftarrow \mathsf{Hash}\big(\mathsf{hk}_j, (\mathsf{hk}_l, \mathsf{EP}_l)_{l \neq j}\big)$
$\forall j \in [n] : \quad (\mathsf{pk}_j, c_j) \leftarrow \mathsf{EP}_j(y_j)$
$C \leftarrow \mathsf{mkFHE.Eval}\big(\tilde{\mathcal{D}}, \mathsf{pk}_1, c_1, \dots, \mathsf{pk}_n, c_n\big)$ (see below)
$(r_i, r'_i, r''_i, \eta_i, \eta'_i) \leftarrow F_2(K_2^{(i)}, y_i)$
$(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{mkFHE.Gen}(1\!\!1^\lambda, i; r_i)$
$d_i \leftarrow \mathsf{mkFHE.PartDec}\Big(C, (\mathsf{pk}_1, \mathsf{pk}_2, \dots, \mathsf{pk}_n), i, \mathsf{sk}_i; \eta_i\Big)$
Output $d_i$
```

The algorithm $\tilde{\mathcal{D}}$. On input n random strings $s_1, s_2, \dots, s_n \in \{0,1\}^{m(\lambda)}$.

```
$s \leftarrow s_1 \oplus s_2 \oplus \dots \oplus s_n$
$R \leftarrow \mathcal{D}(1\!\!1^\lambda; s)$
Output $R$
```

Fig. 29. The unobfuscated decryption program of party P_i

proofs $(\pi_j)_{j \neq i}$ as input and will use them to check the pair $(\mathsf{hk}_j, \mathsf{EP}_j)$ for every $j \neq i$. If any of the NIZKs does not verify, the decryption program DP_i simply outputs $\perp$. We sketch the unobfuscated code of the decryption program DProg in Fig. 29.

Circular dependencies between subexponentially secure primitives. Our construction can achieve security as long as at least one of the random strings s_i remains private. Since the encryption program EP_i always reveals an encryption of the latter, we need to rely on the security of multi-key FHE. Unfortunately, we cannot perform a direct reduction as the PRF key that allows retrieving the multi-key FHE private key is hardcoded into both the encryption and the decryption program. So, in the security proof, we need to somehow remove the information about sk_i from EP_i and DP_i first, and only at that point, we can apply the multi-key FHE security.

Our goal is to achieve this using subexponentially secure primitives, similarly to what Halevi *et al.* did in [HIJ+17]. Specifically, by repeating a hybrid argument for every tuple $(\mathsf{hk}_j, \mathsf{EP}_j)_{j \neq i}$ of well-formed elements, the programs EP_i and DP_i will gradually switch from performing the key generation, the encryptions and the partial decryptions to simulating them [AJJM20]. Notice that the multi-key

FHE simulators need to know the randomness used by all the other parties. The program will extract it from the NIZKs $(\pi_j)_{j\neq i}$ that are given as input.

In order for our strategy to work, we need to rely on the subexponential security of multi-key FHE. In particular, if we denote the number of well-formed tuples $(\mathsf{hk}_j, \mathsf{EP}_j)_{j\neq i}$ by $N(\lambda)$ and the advantage of any PPT adversary $\mathcal{A}$ against the multi-key FHE scheme by $\mathsf{Adv}^{\mathcal{A}}_{\mathsf{mkFHE}}(\lambda)$, we require that there exists a constant $e \in \mathbb{N}$ such that

$$N(\lambda) \cdot \mathsf{Adv}^{\mathcal{A}}_{\mathsf{mkFHE}}(\lambda^e) = \mathsf{negl}(\lambda)^{12}.$$

This is because, every time we rely on the simulatability of the partial decryption, the advantage of the adversary increases by a negligible but non-zero amount. In our proof, we rely on this argument a superpolynomial number of times, namely at least $N(\lambda)$, so, at the end, the small advantages might add up to something non-negligible. If the e described above exists, however, we are sure that, by setting the security parameter of multi-key FHE to $\lambda' := \lambda^e$, this will not happen. The final stage will be indistinguishable from the initial one.

The issue is that $N(\lambda)$ already depends on λ'. Indeed, every encryption program generates a multi-key FHE key. We are therefore trapped in a circular dependency. It also turns out that this is not the only one, it is just the easiest to spot.

Decreasing the entropy of the corrupted messages. We solve our problems using an idea of [ASY22]: we decrease the entropy of $(\mathsf{hk}_j, \mathsf{EP}_j)$ generating it using a PRG. Each party P_j will now sample a random λ-bit seed and will use its expansion to generate the PRF keys hidden in EP_j, the hash key hk_j and to obfuscate EP_j. The NIZK π_j will guarantee that the pair $(\mathsf{hk}_j, \mathsf{EP}_j)$ is generated in this way. In other words, the adversary will be forced to output low-entropy messages. On the other hand, by leveraging the simulatability of the NIZK, we will be able to send full-entropy messages for the honest parties.

Thanks to this trick, the number of well-formed $(\mathsf{hk}_j, \mathsf{EP}_j)_{j\neq i}$ will be independent of the multi-key FHE security parameter λ': the value of $N(\lambda)$ will be $2^{\lambda\cdot(n-1)}$. By choosing e sufficiently large and assuming subexponential security, we can finally make sure that

$$N(\lambda) \cdot \mathsf{Adv}^{\mathcal{A}}_{\mathsf{mkFHE}}(\lambda^e) = \mathsf{negl}(\lambda).$$

This trick fixes all the other circular dependencies too.

Avoiding collisions between well-formed encryption programs. The technique described in the previous paragraph will also allow us to achieve a nice property: by taking a subexponentially collision resistant hash function, we can make sure that, with overwhelming probability over hk_i [13], there exist no hash collisions

[12] To be precise, we will require a strictly stronger property: instead of N, we will use another function $M(\lambda) \gg N(\lambda)$.

[13] The probability is over full-entropy hash keys.

between well-formed tuples $(\mathsf{hk}_j, \mathsf{EP}_j)_{j \neq i}$. In particular, we choose the hash function security parameter λ' so that, for every PPT adversary $\mathcal{A}$,

$$N(\lambda)^2 \cdot \mathsf{Adv}^{\mathcal{A}}_{\mathsf{Hash}}(\lambda') = \mathsf{negl}(\lambda),$$

where $\mathsf{Adv}^{\mathcal{A}}_{\mathsf{Hash}}(\lambda')$ denotes the advantage of $\mathcal{A}$ against the collision resistance of Hash. Notice that $N(\lambda)^2$ upper-bounds the number of pairs of well-formed tuples. This of course will increase the size of the digests but they will still fit into EP_i. We will explain how this property is used in the security proof in Section 7.2.

Adding a final NIZK to achieve active security. The reader might have noticed that in our blueprint, nothing prevents an adversary to broadcast malformed decryption programs. In order to contrast this kind of malicious behaviour, we add a second NIZK to the construction proving the well-formedness of the programs and the hash key. We rely on a simulation-extractable NIZK, we denote it by NIZK'. Observe that the latter satisfied multi-theorem zero-knowledge. If we aim for security against non-uniform adversaries, NIZK' will require a CRS that can be small and uniformly random. In the uniform setting, if we use the construction in Section 9.3, NIZK' has no CRS. We denote the new proof broadcast by party P_i by π'_i. To summarise, the distributed sampler message of P_i will consists of the tuple $U_i := (\mathsf{hk}_i, \mathsf{EP}_i, \mathsf{DP}_i, \pi_i, \pi'_i)$.

7.1 Introducing ELFs to Achieve Lossy Properties

The construction we sketched above is not lossy: given the messages of the honest parties, the output still remains highly unpredictable, i.e. $\mathsf{H}\big(R|(U_i)_{i\in H}\big) = \omega(\log \lambda)$. For this reason, we introduce an ELF in the construction. When the latter is set in injective mode, the entropy of the output given the honest messages will remain high. When the ELF is instead in lossy mode, the messages of the honest players will restrict the output in a set of polynomial size, no matter what the adversary does. The properties of ELFs will also allow us to make the distinguishability advantage between injective mode and lossy mode arbitrarily small. That will be fundamental to achieve the first property of lossy distributed samplers (see Def. 30).

While integrating the ELF in the construction, we need to pay attention to particular conditions. As mentioned in the technical overview, we want the distributed sampler to be regular and programmable: our goal is to hide an ideal sample R among the small set of possible outputs allowed by the lossy mode. Any adversary must have a $1/\mathsf{poly}(\lambda)$ probability of obliviously selecting R as output of the protocol. We need also to focus on incorporating the ELF in the construction while keeping the CRS as simple as possible. Finally, we need to make sure that the protocol supports a polynomial number of parties instead of just a constant.

Where should we place the ELF? Satisfying all the conditions described above is not trivial. Our current construction allows the parties to produce a common string that looks random as long as one party is honest, i.e. the string $s := s_1 \oplus s_2 \oplus \cdots \oplus s_n$. In order to obtain a random looking sample from $\mathcal{D}(\mathbb{1}^\lambda)$, we need a common source of entropy, i.e. entropy that can be accessed by all players. The CRS and s are the only sources of this kind we have at the moment.

After observing this, one could try to achieve the properties we need by feeding s into the ELF, convert the output into uniform randomness using an extractor and then input the result into the distribution $\mathcal{D}(\mathbb{1}^\lambda)$. While adding two CRSs (one for the ELF, one for the extractor), this solution allows to restrict the output in a set of polynomial size when the ELF is set in lossy mode. However, it lacks programmability: how can we hide an ideal sample among the outputs without the adversary noticing it? Observe that, using just the CRSs, the adversary can compute the set of all possible outputs, so the ideal sample would stand out. In cryptography, programmability is often achieved using puncturable PRFs and obfuscation. The technique requires the PRF key K to be private and unpredictable. Hiding K in the CRS seems hard, perhaps even impossible. Another option would be to use s to generate K. The issue is that K needs high entropy, so we cannot use the output of the ELF, we are forced to use s itself. If we do that, however, the size of the output space would become superpolynomial.

$\mathsf{EProg}_{\mathsf{Ls}}[K_2^{(i)}, i]$

Hard-coded. The PPRF key $K_2^{(i)}$ and the index i.
Input. A digest $y \in \{0,1\}^{t(\lambda)}$.

```
1. $(r_i, r'_i, r''_i, \eta_i, \eta'_i) \leftarrow F_2(K_2^{(i)}, y)$
2. $(\phi, \mathsf{pk}_i, c_i) \leftarrow \mathsf{mkFHE.Sim}_1(\mathbb{1}^\lambda, i; r''_i)$
3. Output $(\mathsf{pk}_i, c_i)$.
```

Fig. 30. The unobfuscated encryption program for the lossy mode

The lossy mode of the distributed sampler. We solve all our problems by relying on subexponentially secure primitives and making the ELF appear only when the distributed sampler is set in lossy mode.

The lossy mode will produce programs EP_i and DP_i that differ from the ones in standard mode, we formally describe them in Fig. 30 and Fig. 31. The idea is that EP_i and DP_i will simulate the key generation, the encryptions and the partial decryptions. The security of multi-key FHE will guarantee that the output of the distributed sampler will be the sample given to the multi-key FHE simulator. Such sample will not coincide with the value hidden in the evaluated ciphertext C, it will be the element produced by $\mathsf{Project}$ and $\mathsf{Extract}$. The former will simply apply an ELF f on $(\mathsf{hk}_j, \mathsf{EP}_j)_{j \in [n]}$. The latter will use a puncturable

PRF key K to deterministically map each projection into a pseudorandom string s, which will be used as randomness for $\mathcal{D}(\mathbb{1}^\lambda)$. Notice that since the ELF is in lossy mode, the image of the projection will be polynomial in size.

Working out the rest of the details is now easy. The decryption program will be able to run $\mathsf{Project}$ and $\mathsf{Extract}$ inside its code as f and K will be hardcoded into it. It will also be able to perform the partial decryption as it will know the PRF keys hardcoded in the encryption programs that are given as input: it will extract them from the NIZKs that are provided along with $(\mathsf{EP}_j)_{j \neq i}$. When the extraction fails, the program will simply output $\perp$. In order for our strategy to succeed, the lossy setup will simulate the CRSs σ and σ'. The corresponding trapdoors will also allow us to generate proofs π_i and π'_i despite the fact that EP_i and DP_i are no longer well-formed. The lossy setup will also take care of generating the ELF f. The size of the image will be $q(\lambda)$ where q is the polynomial parametrising the lossy mode of the distributed sampler. A final minor issue is that the second multi-key FHE simulator needs to receive the state produced by the first simulator. The execution of the former takes place in DP_i, whereas the latter is run in EP_i. Thankfully, both executions are made deterministic using the outputs of a puncturable PRF. By storing the corresponding key in both EP_i and DP_i, we can rerun the first simulator inside DP_i to retrieve the state.

Regularity and programmability of the lossy mode. The above construction can easily be made regular. It is sufficient to use a regular ELF: by sampling a random element x in the domain $[M]$, $f(x)$ will hit all the elements in the support of the projection with inverse-polynomial probability.

The construction is also programmable. Thanks to obfuscation and the security of puncturable PRFs [SW14], we can easily hide an ideal sample in the output space of the lossy mode distributed sampler. All we need to do is to puncture K in the right position z. We then modify the decryption program by hardcoding an ideal sample R along with z and the punctured key. Differently from before, the new program will compare the output of the ELF with z. When the latter coincide, it will directly feed R to the partial decryption simulator. It is easy to prove that the adversary is not able to detect whether we hid an ideal sample in the output space or not.

7.2 Proving Security

We present a blueprint of the security proof. Seeing that, in our construction, the projection has small support is rather straightforward. We therefore focus on the first property of the lossy distributed sampler. The proof will hold independently of whether AClass represents the class of uniform adversaries or not.

Before starting, we recall our goal: we want to show that for every polynomial $p(\lambda)$ and inverse-polynomial function $\delta(\lambda)$, there exists a polynomial $q(\lambda)$ such that the advantage of all adversaries running in time at most $p(\lambda)$ in distinguishing the standard mode from the lossy mode parametrised by $q(\lambda)$ is asymptotically smaller than $\delta(\lambda)$. We prove the result through a series of hybrids, starting from the standard mode.

$\mathsf{DProg}_{\mathsf{Ls}}[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j \neq i}, K, f]$

Hard-coded. The index i of the party, the session identity sid, a PPRF key $K_2^{(i)}$, the encryption program EP_i, the hash key hk_i, the extractable NIZK CRS σ and the extraction trapdoors $(\tau_e^j)_{j \neq i}$, the PPRF key K, the ELF f.
Input. Set of $n-1$ tuples $(\mathsf{hk}_j, \mathsf{EP}_j, \pi_j)_{j \neq i}$.

```
 1. $\forall j \neq i: \quad b_j \leftarrow \mathsf{NIZK.Verify}\big(\sigma, (\mathsf{sid}, j), \pi_j, (j, \mathsf{hk}_j, \mathsf{EP}_j)\big)$
 2. $\forall j \neq i: \quad \big(K_1^{(j)}, K_2^{(j)}\big) \leftarrow \mathsf{NIZK.Extract}\big(\tau_e^j, \pi_j, (j, \mathsf{hk}_j, \mathsf{EP}_j)\big)$ [a]
 3. If $\exists j \neq i$ such that $b_j = 0$ or $\big(K_1^{(j)}, K_2^{(j)}\big) = \bot$, output $\bot$
 4. $\forall j \in [n]: \quad y_j \leftarrow \mathsf{Hash}\big(\mathsf{hk}_j, (\mathsf{hk}_l, \mathsf{EP}_l)_{l \neq j}\big)$
 5. $\forall j \neq i: \quad s_j \leftarrow F_1\big(K_1^{(j)}, y_j\big)$
 6. $\forall j \in [n]: \quad (r_j, r_j', r_j'', \eta_j, \eta_j') \leftarrow F_2\big(K_2^{(j)}, y_j\big)$
 7. $z \leftarrow f\big((\mathsf{hk}_j, \mathsf{EP}_j)_{j \in [n]}\big)$
 8. $s \leftarrow F(K, z)$
 9. $\hat{R} \leftarrow \mathcal{D}(1\!\!1^\lambda; s)$
10. $(\phi, \mathsf{pk}_i, c_i) \leftarrow \mathsf{mkFHE.Sim}_1(1\!\!1^\lambda, i; r_i'')$
11. $d_i \leftarrow \mathsf{mkFHE.Sim}_2\Big(\phi, \tilde{\mathcal{D}}, \hat{R}, (s_j, r_j, r_j')_{j \neq i}; \eta_i'\Big)$ (see bottom of Fig. 29)
12. Output $d_i$
```

[a] Here, we simplified the notation: the extractor would output only the PRG seed used to produce $(\mathsf{hk}_j, \mathsf{EP}_j)$. By the expanding that, it is straightforward to derive $K_1^{(j)}$ and $K_2^{(j)}$.

Fig. 31. The unobfuscated decryption program for the lossy mode

First step: simulating NIZK'. We start the proof by simulating the proof π_i' in every $\mathsf{NewSession}$ query. We recall that i denotes the index chosen by the adversary in each of these queries, π_i' denotes instead the simulation-extractable NIZK proving the well-formedness of the message of i-th party. We also modify the answer to the sampling queries: we start by extracting the witnesses from the NIZKs $(\pi_j')_{j \neq i}$ selected by the adversary. If the extraction fails, we answer with $\bot$, otherwise, we reply with the output of Sample. This hybrid is indistinguishable from the previous one due to the simulation-extractability of NIZK'.

Second step: witness extraction in the decryption programs. We proceed by simulating the proof π_i in every $\mathsf{NewSession}$ query. We recall that π_i is an almost everywhere extractable NIZK proving the well-fomredness of $(\mathsf{hk}_i, \mathsf{EP}_i)$. We also modify the decryption program DP_i. Specifically, we hardcode extraction trapdoors $(\tau_e^j)_{j \neq i}$ for the almost everywhere extractable NIZK. The label associated with τ_e^j will be (sid, j) where sid is the session identity queried by the adversary. The program DP_i will now try to extract the witness from the NIZK proofs that are given as input. If any extraction fails, DP_i will simply outputs $\bot$. Otherwise, it will perform the usual operations. Notice that now the decryption program

$\mathsf{DProg}_1[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j \neq i}]$

Hard-coded. The index i of the party, the session identity sid, a PPRF key $K_2^{(i)}$, the encryption program EP_i, the hash key hk_i, the CRS for the extractable NIZK σ, the extraction trapdoors $(\tau_e^j)_{j \neq i}$.
Input. Set of $n-1$ tuples $(\mathsf{hk}_j, \mathsf{EP}_j, \pi_j)_{j \neq i}$.

```
$\forall j \neq i :\quad b_j \leftarrow \mathsf{NIZK.Verify}\big(\sigma, (\mathsf{sid}, j), \pi_j, (j, \mathsf{hk}_j, \mathsf{EP}_j)\big)$
$\forall j \neq i :\quad \big(K_1^{(j)}, K_2^{(j)}\big) \leftarrow \mathsf{NIZK.Extract}\big(\tau_e^j, \pi_j, (j, \mathsf{hk}_j, \mathsf{EP}_j)\big)^{a}$
If $\exists j \neq i$ such that $b_j = 0$ or $\big(K_1^{(j)}, K_2^{(j)}\big) = \bot$, output $\bot$
$\forall j \in [n] :\quad y_j \leftarrow \mathsf{Hash}\big(\mathsf{hk}_j, (\mathsf{hk}_l, \mathsf{EP}_l)_{l \neq j}\big)$
$\forall j \in [n] :\quad (\mathsf{pk}_j, c_j) \leftarrow \mathsf{EP}_j(y_j)$
$C \leftarrow \mathsf{mkFHE.Eval}\big(\tilde{\mathcal{D}}, \mathsf{pk}_1, c_1, \ldots, \mathsf{pk}_n, c_n\big)$ (see Fig. 29)
$(r_i, r_i', r_i'', \eta_i, \eta_i') \leftarrow F_2(K_2^{(i)}, y_i)$
$(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{mkFHE.Gen}(1\!\!1^\lambda, i;\, r_i)$
$d_i \leftarrow \mathsf{mkFHE.PartDec}\Big(C, (\mathsf{pk}_1, \mathsf{pk}_2, \ldots, \mathsf{pk}_n), i, \mathsf{sk}_i;\, \eta_i\Big)$
Output $d_i$
```

Fig. 32. Second step: the unobfuscated decryption program of party P_i

will only accept well-formed inputs. We sketch the operations of the modified program DProg_1 in Fig. 32.

We highlight that, compared to the previous step, the input-output behaviour of DP_i changed. However, the two hybrids will still be indistinguishable thanks to the almost-everywhere extractability of NIZK (see Lemma 2 and Lemma 3). In the uniform setting, this step requires additional attention. Indeed, in the reduction to almost-everywhere extractability, the uniform adversary needs to derive the trapdoor τ' for NIZK'. The latter cannot be computed in uniform polynomial time. We work around this problem by choosing NIZK, NIZK' and a superpolynomial function $S(\lambda)$ so that NIZK' is S-deterministic and NIZK is a-disclosed for every S-computable sequence a. The construction presented in Section 10.1 allows this.

Third step: switching to full-entropy. Since the NIZKs are now simulated, we are free to switch to a full-entropy U_i. Specifically, we generate the hash key hk_i, the encryption program EP_i and the keys hardcoded into it using full-entropy randomness, instead of the output of a PRG. This stage is indistinguishable from the previous by the security of the PRG.

Fourth step: simulating key generation, ciphertexts and partial decryptions. We proceed by modifying both the encryption program EP_i and the decryption program DP_i. The new programs will not perform the multi-key FHE operations as usual, they will instead simulate them. In order to perform such operation, DP_i will use the PRF keys in the encryption programs of the other parties,

$\mathsf{DProg}_2[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j\neq i}, K_1^{(i)}]$

Hard-coded. The index i of the party, the session identity sid, a PPRF key $K_2^{(i)}$, the encryption program EP_i, the hash key hk_i, the extractable NIZK CRS σ and the extraction trapdoors $(\tau_e^j)_{j\neq i}$, the PPRF key $K_1^{(i)}$.
Input. Set of $n-1$ tuples $(\mathsf{hk}_j, \mathsf{EP}_j, \pi_j)_{j\neq i}$.

```
$\forall j \neq i:$   $b_j \leftarrow \mathsf{NIZK.Verify}\big(\sigma, (\mathsf{sid}, j), \pi_j, (j, \mathsf{hk}_j, \mathsf{EP}_j)\big)$
$\forall j \neq i:$   $\big(K_1^{(j)}, K_2^{(j)}\big) \leftarrow \mathsf{NIZK.Extract}\big(\tau_e^j, \pi_j, (j, \mathsf{hk}_j, \mathsf{EP}_j)\big)$ [a]
If $\exists j \neq i$ such that $b_j = 0$ or $\big(K_1^{(j)}, K_2^{(j)}\big) = \bot$, output $\bot$
$\forall j \in [n]:$   $y_j \leftarrow \mathsf{Hash}\big(\mathsf{hk}_j, (\mathsf{hk}_l, \mathsf{EP}_l)_{l\neq j}\big)$
$\forall j \in [n]:$   $s_j \leftarrow F_1\big(K_1^{(j)}, y_j\big)$
$\forall j \in [n]:$   $(r_j, r'_j, r''_j, \eta_j, \eta'_j) \leftarrow F_2\big(K_2^{(j)}, y_j\big)$
$\hat{R} \leftarrow \mathcal{D}(1\!\!1^\lambda; s_1 \oplus \cdots \oplus s_n)$
$(\phi, \mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{mkFHE.Sim}_1(1\!\!1^\lambda, i; r''_i)$
$d_i \leftarrow \mathsf{mkFHE.Sim}_2\Big(\phi, \tilde{\mathcal{D}}, \hat{R}, (s_j, r_j, r'_j)_{j\neq i}; \eta'_i\Big)$ (see bottom of Fig. 29)
Output $d_i$
```

Fig. 33. Forth step: the unobfuscated decryption program of party P_i

which will be extracted from the NIZKs $(\pi_j)_{j\neq i}$ that are given as input. For the moment, the multi-key FHE simulator in DP_i will also need to know the sample $\hat{R}$ hidden into the joint ciphertext C. The program will reconstruct it using the PRF keys $(K_1^{(j)})_{j\neq i}$ hidden in the encryption programs $(\mathsf{EP}_j)_{j\neq i}$, those used to produce the pseudorandom strings $(s_j)_{j\neq i}$. In order to compute s_i, the new program will also have the key $K_1^{(i)}$ hardcoded. Finally, the simulator in DP_i will need to know that secret information output by the first simulator Sim_1, the one that produced a fake public key and a fake ciphertext in EP_i. The decryption program will obtain it by rerunning Sim_1 with the same randomness as in EP_i. It is easy to do that as the randomness is a PRF output and the corresponding key is hardcoded in both EP_i and DP_i. We sketch the unobfuscated version of the modified programs $\mathsf{EProg}_{\mathsf{Ls}}$ and DProg_2 in Fig. 30 and Fig. 33.

We prove that this step is indistinguishable from the previous one using an exponential number of hybrids. In particular, the number of reductions is proportional to the number of digests of the hash function, i.e. $2^{t(\lambda)}$. The proof relies on the reusable semi-malicious security of multi-key FHE, the collision resistance of the hash function, the security of iO and the one of the puncturable PRF F_2, all four subexponential. In order for the proof to go through, we need to use an injective obfuscator. In this way, we are sure that EP_j uniquely determines the PRF keys $K_1^{(j)}$ and $K_2^{(j)}$, so the NIZK extraction will always lead to the same values.

```
THE STANDARD MODE OF THE LOSSY DISTRIBUTED SAMPLER
$\mathsf{Setup}(\mathbb{1}^\lambda)$:

  1. $\sigma \xleftarrow{\$} \mathsf{NIZK.Gen}(\mathbb{1}^\lambda)$
  2. $\sigma' \xleftarrow{\$} \mathsf{NIZK'.Gen}(\mathbb{1}^\lambda)$
  3. Output $\mathsf{crs} := (\sigma, \sigma')$

$\mathsf{Gen}\big(\mathbb{1}^\lambda, \mathsf{sid}, i, \mathsf{crs} := (\sigma, \sigma')\big)$:

  1. $\rho_1 \xleftarrow{\$} \{0,1\}^{L_1(\lambda)}$
  2. $\rho_2 \xleftarrow{\$} \{0,1\}^{L_2(\lambda)}$
  3. $W \xleftarrow{\$} \{0,1\}^\lambda$
  4. $(K_1^{(i)}, K_2^{(i)}, u_1, u_2) \leftarrow \mathsf{PRG}(W)$
  5. $\mathsf{hk}_i \leftarrow \mathsf{Hash.Gen}(\mathbb{1}^\lambda;\, u_1)$
  6. $\mathsf{EP}_i \leftarrow \mathsf{iO}(\mathbb{1}^\lambda, \mathsf{EProg}[K_1^{(i)}, K_2^{(i)}, i];\, u_2)$ (see Fig. 28)
  7. $\mathsf{DP}_i \leftarrow \mathsf{iO}(\mathbb{1}^\lambda, \mathsf{DProg}[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma];\, \rho_1)$ (see Fig. 29)
  8. $\pi_i \leftarrow \mathsf{NIZK.Prove}\big(\sigma, (\mathsf{sid}, i), (i, \mathsf{hk}_i, \mathsf{EP}_i), W;\, \rho_2\big)$
  9. $\pi'_i \xleftarrow{\$} \mathsf{NIZK'.Prove}\big(\sigma', (i, \mathsf{sid}, \mathsf{hk}_i, \mathsf{EP}_i, \mathsf{DP}_i, \pi_i, \sigma), (W, \rho_1, \rho_2)\big)$
 10. Output $U_i := (\mathsf{hk}_i, \mathsf{EP}_i, \mathsf{DP}_i, \pi_i, \pi'_i)$.

$\mathsf{Sample}\Big(\big(U_j = (\mathsf{hk}_j, \mathsf{EP}_j, \mathsf{DP}_j, \pi_j, \pi'_j)\big)_{j\in[n]}, \mathsf{sid}, \mathsf{crs} = (\sigma, \sigma')\Big)$

  1. $\forall j \in [n]:\quad b_j \leftarrow \mathsf{NIZK'.Verify}\big(\sigma', \pi'_j, (j, \mathsf{sid}, \mathsf{hk}_j, \mathsf{EP}_j, \mathsf{DP}_j, \pi_j, \sigma)\big)$
  2. If there exists $j \in [n]$ such that $b_j = 0$, output $\perp$.
  3. $\forall j \in [n]:\quad d_j \leftarrow \mathsf{DP}_j\big((\mathsf{hk}_l, \mathsf{EP}_l, \pi_l)_{l\neq j}\big)$
  4. $R \leftarrow \mathsf{mkFHE.FinDec}(d_1, \dots, d_n)$
  5. Output $R$
```

Fig. 34. The standard mode of the lossy distributed sampler

Fifth step: embedding the ELF into the decryption program DP_i*.* In this step, we finally integrate the ELF in the construction. For the moment, the ELF will be set in injective mode.

We observe that at the end of step four, we have finally managed to unlink $K_1^{(i)}$ from $(s_j)_{j\neq i}$. Previously, indeed, it was impossible to modify $K_1^{(i)}$ without affecting $(s_j)_{j\neq i}$. By changing $K_1^{(i)}$, the encryption program EP_i would have become different, consequently all the digests $(y_j)_{j\neq i}$ would have changed and, at the end, we would end up with new PRF outputs $(s_j)_{j\neq i}$. Since all the strings $s_1, \dots, s_n$ where mutually dependent, it was hard to analyse how the adversary could affect the distribution of their sum. Now, instead, by the security of the puncturable PRF, we can finally say that s_i looks independent of $(s_j)_{j\neq i}$. So, we are sure that our construction generates pseudorandom samples.

We leverage this fact to modify the decryption program DP_i once again, switching to an obfuscation of $\mathsf{DProg}_{\mathsf{Ls}}$ (see Fig. 31). The new program will ignore $(s_j)_{j\neq i}$. It will instead feed the inputs $(\mathsf{hk}_j, \mathsf{EP}_j)_{j\neq i}$ along with the hardcoded pair $(\mathsf{hk}_i, \mathsf{EP}_i)$ into the injective-mode ELF. The result is then input into a puncturable PRF. The randomness produced in this way is given to $\mathcal{D}(\mathbb{1}^\lambda)$. The generated sample will be input into the partial decryption simulator. We select the ELF so that its domain is sufficiently large to embed all $(\mathsf{hk}_j, \mathsf{EP}_j)_{j\in[n]}$ into it without causing any collision. In conclusion, in the new program, each tuple $(\mathsf{hk}_j, \mathsf{EP}_j)_{j\neq i}$ will be mapped to an independent-looking pseudorandom output.

Using a superpolynomial number of hybrids, we prove that step five and step four are indistinguishable. In particular, we repeat the hybrid arguments for every well-formed tuple $(\mathsf{hk}_j, \mathsf{EP}_j)_{j\neq i}$, i.e. $2^{\lambda\cdot(n-1)}$ times. Observe that one way the adversary can try to distinguish between step four and step five is by finding two pairs of inputs $(\mathsf{hk}_j, \mathsf{EP}_j)_{j\neq i}$, $(\mathsf{hk}'_j, \mathsf{EP}'_j)_{j\neq i}$ having colliding digests under hk_i. Indeed, in step four, the two inputs would produce the same string s_i and therefore, the respective outputs would be correlated. In step five, instead, the adversary would end up with independent looking outputs. We prevent this attack by relying on the subexponential collision intractability of the hash function so that, with overwhelming probability over hk_i, there exist no collisions among the $2^{\lambda\cdot(n-1)}$ well-formed tuples $(\mathsf{hk}_j, \mathsf{EP}_j)_{j\neq i}$. To summarise, we prove indistinguishability between step four and five by relying on the security of iO, the collision intractability of the hash function and the security of the puncturable PRFs F and F_1, all of them subexponential.

Final step: setting the ELF in lossy mode. At this point, we switch the ELF hidden in DP_i into lossy mode. Notice that step six can be distinguished from step five, however, by properly setting the parameters of the lossy mode, we can make the distinguishing advantage arbitrarily small. In particular, let $p'(\lambda)$ be the total time needed by the challenger and the adversary in step five. We can pick the polynomial $q(\lambda)$ parametrising the lossy mode so that no adversary running in time $p'(\lambda)$ can distinguish between injective mode and lossy mode with advantage greater than $\delta/2$. In this way, we are sure that the distinguishability advantage between step zero and step six is at most $\delta/2 + \mathsf{negl}(\lambda)$. This step corresponds to the lossy mode of the distributed sampler.

7.3 Formalising the Results

The full description of the standard mode of our distributed sampler is in Fig. 34. In the construction, NIZK denotes an almost everywhere extractable NIZK. When we aim for security against non-uniform adversaries, NIZK will be chosen-ID zero-knowledge and almost everywhere extractable as in Def. 17. In the uniform setting instead, we will rely on simulation almost-everywhere extractability and zero-knowledge as in Def. 46. The NP relation underlying NIZK is

$$\mathcal{R}_1 := \left\{ \begin{matrix}(i, \mathsf{hk}_i, \mathsf{EP}_i),\\ W\end{matrix} \;\middle|\; \begin{aligned}&(K_1^{(i)}, K_2^{(i)}, u_1, u_2) := \mathsf{PRG}(W)\\ &\mathsf{hk}_i = \mathsf{Hash.Gen}(\mathbb{1}^\lambda;\, u_1)\\ &\mathsf{EP}_i = \mathsf{iO}(\mathbb{1}^\lambda, \mathsf{EProg}[K_1^{(i)}, K_2^{(i)}, i];\, u_2)\end{aligned} \right\}$$

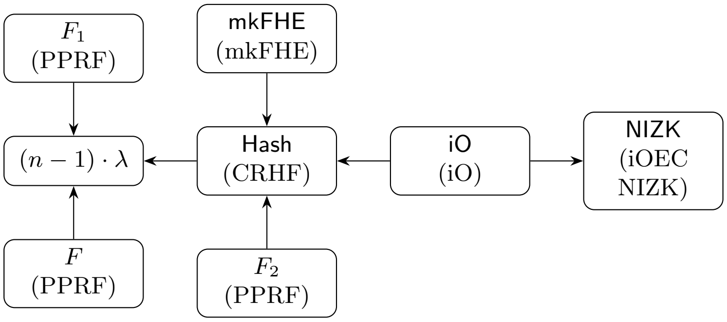


Fig. 35. In this diagram, we describe the dependencies between the security parameters of the various subexponentially secure primitives. When a primitive is connected through an arrow to $(n-1)\cdot\lambda$, we mean that the advantage of any PPT adversary against the security of the primitive must be $\mathsf{negl}(\lambda)/2^{(n-1)\cdot\lambda}$. When a primitive is connected to Hash, we mean that the advantage of any PPT adversary against the security of the primitive must be $\mathsf{negl}(\lambda)/2^{t(\lambda)}$. We recall that $t(\lambda)$ denotes the length of the digests. When a primitive is connected to NIZK, we mean that the advantage of any PPT adversary against the security of the primitive must be $\mathsf{negl}(\lambda)/d(\lambda)$ where $d(\lambda)$ is the upper-bound on $|\mathsf{VPFE}_{\sigma,\tau_e}|$ in NIZK.

We also make use of a simulation-extractable NIZK, which we will denote by NIZK'. In the uniform setting, NIZK' will be $S(\lambda)$-deterministic whereas NIZK will be a-compatible for every $S(\lambda)$-computable sequence a. In other words, NIZK will be secure even if we leak the trapdoor τ' for NIZK'. The relation corresponding to NIZK' is the following.

$$\mathcal{R}_2 := \left\{ \begin{array}{c|l} \big((i,\mathsf{sid},\mathsf{hk}_i,\mathsf{EP}_i, & (K_1^{(i)},K_2^{(i)},u_1,u_2) := \mathsf{PRG}(W) \\ \mathsf{DP}_i,\pi_i,\sigma), & \big((i,\mathsf{hk}_i,\mathsf{EP}_i),W\big)\in\mathcal{R}_1 \\ (W,\rho_1,\rho_2)\big) & \mathsf{DP}_i = \mathsf{iO}(\mathbb{1}^\lambda,\mathsf{DProg}[i,\mathsf{sid},K_2^{(i)},\mathsf{EP}_i,\mathsf{hk}_i,\sigma];\ \rho_1) \\ & \pi_i = \mathsf{NIZK.Prove}\big(\sigma,(\mathsf{sid},i),(i,\mathsf{hk}_i,\mathsf{EP}_i),W;\ \rho_2\big) \end{array} \right\}$$

Above, we used L_1 and L_2 to denote the length of the randomness used to obfuscate DProg and to prove a statement using NIZK, respectively.

We rely on an injective and subexponentially secure indistinguishability obfuscator iO. We also use a multi-key FHE scheme mkFHE that satisfies subexponential reusable semi-malicious security. Let Hash be a subexponentially collision resistant hash function, outputting digests of length $t(\lambda)$. We use two subexponentially secure puncturable PRFs F_1 and F_2. The first one outputs a pseudorandom string of length equal to the randomness needed by $\mathcal{D}(\mathbb{1}^\lambda)$. The second one outputs a pseudorandom string of length equal to the randomness needed by $\mathsf{mkFHE.Gen}$, $\mathsf{mkFHE.Enc}$, $\mathsf{mkFHE.Sim}_1$, $\mathsf{mkFHE.PartDec}$ and $\mathsf{mkFHE.Sim}_2$. Finally, we rely on a PRG mapping a λ-bit seed W into a pseudorandom string (K_1,K_2,u_1,u_2) where K_1 and K_2 are PRF keys for F_1 and F_2 respectively and

u_1 and u_2 are as long as the randomness needed by $\mathsf{Hash.Gen}$ and the obfuscation of EProg respectively.

```
THE LOSSY MODE OF THE DISTRIBUTED SAMPLER
$\mathsf{LossySetup}\big(\mathbb{1}^\lambda, q(\lambda)\big)$:

  1. $(\sigma, \tau_s, \tau_e) \xleftarrow{\$} \mathsf{NIZK.SimSetup}(\mathbb{1}^\lambda)$
  2. $(\sigma', \tau') \xleftarrow{\$} \mathsf{NIZK'.SimSetup}(\mathbb{1}^\lambda)$
  3. $f \xleftarrow{\$} \mathsf{ELF.Gen}(M, q)$
  4. Output $\mathsf{crs} := (\sigma, \sigma')$ and $\zeta := (\sigma, \sigma', \tau_s, \tau_e, \tau', f)$.

$\mathsf{LossyGen}\big(\mathbb{1}^\lambda, \mathsf{sid}, i, \zeta := (\sigma, \sigma', \tau_s, \tau_e, \tau', f)\big)$:

  1. $K \xleftarrow{\$} F.\mathsf{Gen}(\mathbb{1}^\lambda)$
  2. $K_2^{(i)} \xleftarrow{\$} F_2.\mathsf{Gen}(\mathbb{1}^\lambda)$
  3. $\mathsf{hk}_i \xleftarrow{\$} \mathsf{Hash.Gen}(\mathbb{1}^\lambda)$
  4. $\mathsf{EP}_i \xleftarrow{\$} \mathsf{iO}(\mathbb{1}^\lambda, \mathsf{EProg}_{\mathsf{Ls}}[K_2^{(i)}, i])$ (see Fig. 30)
  5. $\forall j \neq i: \quad \tau_e^j \xleftarrow{\$} \mathsf{NIZK.Trap}\big(\tau_e, (\mathsf{sid}, j)\big)$
  6. $\mathsf{DP}_i \xleftarrow{\$} \mathsf{iO}(\mathbb{1}^\lambda, \mathsf{DProg}_{\mathsf{Ls}}[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j\neq i}, K, f])$ (see Fig. 31)
  7. $\pi_i \xleftarrow{\$} \mathsf{NIZK.SimProve}\big(\tau_s, (\mathsf{sid}, i), (i, \mathsf{hk}_i, \mathsf{EP}_i)\big)$
  8. $\pi'_i \xleftarrow{\$} \mathsf{NIZK'.SimProve}\big(\tau', (i, \mathsf{sid}, \mathsf{hk}_i, \mathsf{EP}_i, \mathsf{DP}_i, \pi_i, \sigma)\big)$
  9. Output $U_i := (\mathsf{hk}_i, \mathsf{EP}_i, \mathsf{DP}_i, \pi_i, \pi'_i)$ and $\xi_e := K$.

$\mathsf{Project}\Big(\zeta = (\sigma, \sigma', \tau_s, \tau_e, \tau', f), \big(U_j = (\mathsf{hk}_j, \mathsf{EP}_j, \mathsf{DP}_j, \pi_j, \pi'_j)\big)_{j\in[n]}, \mathsf{sid}\Big)$:

  1. $\forall j \in [n]: \quad b_j \leftarrow \mathsf{NIZK'.Verify}\big(\sigma', \pi'_j, (j, \mathsf{sid}, \mathsf{hk}_j, \mathsf{EP}_j, \mathsf{DP}_j, \pi_j, \sigma)\big)$
  2. If there exists $j \in [n]$ such that $b_j = 0$, output $\bot$.
  3. Output $f\big((\mathsf{hk}_j, \mathsf{EP}_j)_{j\in[n]}\big)$.

$\mathsf{Extract}(\xi_e = K, z)$:

  1. If $z = \bot$, output $\bot$.
  2. $s \leftarrow F(K, z)$
  3. Output $\mathcal{D}(\mathbb{1}^\lambda; s)$.
```

Fig. 36. The lossy mode of the distributed sampler

In Fig. 36, we formalise the lossy mode of the distributed sampler. Notice that the construction relies on a subexponentially secure puncturable PRF F. Its outputs are pseudorandom strings that are as long as the randomness needed by $\mathcal{D}(\mathbb{1}^\lambda)$. The construction relies also an ELF. We choose the domain of the latter so that all tuples $(\mathsf{hk}_j, \mathsf{EP}_j)_{j\in[n]}$ fit into it. In Fig. 37, we present the algorithms used for programmability and regularity. We describe the dependencies between the subexponentially secure primitives in Fig. 35.

```
PROGRAMMABILITY AND REGULARITY OF THE LOSSY DISTRIBUTED SAMPLER
$\mathsf{ProgGen}\big(\mathbb{1}^\lambda, \mathsf{sid}, i, z, R, \zeta := (\sigma, \sigma', \tau_s, \tau_e, \tau', f)\big)$:

  1. $K \xleftarrow{\$} F.\mathsf{Gen}(\mathbb{1}^\lambda)$
  2. $K^* \leftarrow F.\mathsf{Punct}(K, z)$
  3. $K_2^{(i)} \xleftarrow{\$} F_2.\mathsf{Gen}(\mathbb{1}^\lambda)$
  4. $\mathsf{hk}_i \xleftarrow{\$} \mathsf{Hash.Gen}(\mathbb{1}^\lambda)$
  5. $\mathsf{EP}_i \xleftarrow{\$} \mathsf{iO}(\mathbb{1}^\lambda, \mathsf{EProg}_{\mathsf{Ls}}[K_2^{(i)}, i])$ (see Fig. 30)
  6. $\forall j \neq i :\quad \tau_e^j \xleftarrow{\$} \mathsf{NIZK.Trap}\big(\tau_e, (\mathsf{sid}, j)\big)$
  7. $\mathsf{DP}_i \xleftarrow{\$} \mathsf{iO}(\mathbb{1}^\lambda, \mathsf{DProg}_{\mathsf{Pr}}[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j\neq i}, K^*, z, f, R])$ (see Fig. 38)
  8. $\pi_i \xleftarrow{\$} \mathsf{NIZK.SimProve}\big(\tau_s, (\mathsf{sid}, i), (i, \mathsf{hk}_i, \mathsf{EP}_i)\big)$
  9. $\pi'_i \xleftarrow{\$} \mathsf{NIZK'.SimProve}\big(\tau', (i, \mathsf{sid}, \mathsf{hk}_i, \mathsf{EP}_i, \mathsf{DP}_i, \pi_i, \sigma)\big)$
 10. Output $U_i := (\mathsf{hk}_i, \mathsf{EP}_i, \mathsf{DP}_i, \pi_i, \pi'_i)$.

$\mathcal{Z}\big(\zeta = (\sigma, \sigma', \tau_s, \tau_e, \tau', f)\big)$:

  1. $b \xleftarrow{\$} \{0, 1\}$
  2. If $b = 0$, output $\perp$.
  3. $x \xleftarrow{\$} [M]$
  4. Output $f(x)$.
```

Fig. 37. Programmability and regularity of the lossy distributed sampler

Theorem 16. *Assume the existence of ELFs and the subexponential security of injective iO, multi-key FHE, collision resistant hash functions and puncturable PRFs. If* AClass *denotes the class of non-uniform adversaries, we also assume the existence of simulation-extractable NIZKs and almost everywhere extractable NIZKs with unstructured CRS. If instead* AClass *denotes the class of uniform adversaries, we additionally assume the existence of simulation-extractable NIZKs and simulation almost everywhere extractable NIZKs with no CRS.*

Then, the construction in Fig. 34 is a programmable lossy distributed sampler for $\mathcal{D}(\mathbb{1}^\lambda)$ *with security against* AClass. *If the ELF is regular, the lossy distributed sampler is also regular. If* AClass *denotes the class of non-uniform adversaries, the construction relies on an unstructured CRS whose size is independent of* $\mathcal{D}(\mathbb{1}^\lambda)$. *If* AClass *denotes the class on uniform adversaries, the construction does not need any CRS.*

Moreover, let $p'(\lambda)$ *denote a polynomial upper-bounding the running times of* $\mathsf{LossySetup}$, $\mathsf{LossyGen}$ *and* $\mathsf{LossySample}$. *The advantage of an adversary* $\mathcal{A}$ *running in time at most* $p(\lambda)$ *in distinguishing between the lossy mode and the standard mode is*

$$\mathsf{Adv}^{M,q}_{\mathsf{ELF},\mathcal{A}'}(\lambda) + \mathsf{negl}(\lambda),$$

$\mathsf{DProg}_{\mathsf{Pr}}[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j\neq i}, K^*, \hat{z}, f, R]$

Hard-coded. The index i of the party, the session identity sid, a PPRF key $K_2^{(i)}$, the encryption program EP_i, the hash key hk_i, the extractable NIZK CRS σ and the extraction trapdoors $(\tau_e^j)_{j\neq i}$, a punctured PRF key K^*, the position $\hat{z}$, the ELF f, the ideal sample R.
Input. Set of $n-1$ tuples $(\mathsf{hk}_j, \mathsf{EP}_j, \pi_j)_{j\neq i}$.

```
$\forall j \neq i: \quad b_j \leftarrow \mathsf{NIZK.Verify}\big(\sigma, (\mathsf{sid}, j), \pi_j, (j, \mathsf{hk}_j, \mathsf{EP}_j)\big)$
$\forall j \neq i: \quad \big(K_1^{(j)}, K_2^{(j)}\big) \leftarrow \mathsf{NIZK.Extract}\big(\tau_e^j, \pi_j, (j, \mathsf{hk}_j, \mathsf{EP}_j)\big)^a$
If $\exists j \neq i$ such that $b_j = 0$ or $\big(K_1^{(j)}, K_2^{(j)}\big) = \perp$, output $\perp$
$\forall j \in [n]: \quad y_j \leftarrow \mathsf{Hash}\big(\mathsf{hk}_j, (\mathsf{hk}_l, \mathsf{EP}_l)_{l\neq j}\big)$
$\forall j \neq i: \quad s_j \leftarrow F_1\big(K_1^{(j)}, y_j\big)$
$\forall j \in [n]: \quad (r_j, r'_j, r''_j, \eta_j, \eta'_j) \leftarrow F_2\big(K_2^{(j)}, y_j\big)$
$z \leftarrow f\big((\mathsf{hk}_j, \mathsf{EP}_j)_{j\in[n]}\big)$
$s \leftarrow F(K^*, z)$
$\hat{R} \leftarrow \mathcal{D}(1\!\!1^\lambda; s)$
If $z = \hat{z}$, $\hat{R} \leftarrow R$
$(\phi, \mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{mkFHE.Sim}_1(1\!\!1^\lambda, i; r''_i)$
$d_i \leftarrow \mathsf{mkFHE.Sim}_2\Big(\phi, \tilde{\mathcal{D}}, \hat{R}, (s_j, r_j, r'_j)_{j\neq i}; \eta'_i\Big)$ (see bottom of Fig. 29)
Output $d_i$
```

Fig. 38. The unobfuscated decryption program for programmability

where $\mathcal{A}'$ is an adversary running in time at most $p(\lambda)^2 \cdot p'(\lambda)$ and $\mathsf{Adv}^{M,q}_{\mathsf{ELF},\mathcal{A}'}(\lambda)$ denotes the advantage of $\mathcal{A}'$ in distinguishing between the injective mode of the ELF with domain size M and its lossy mode parametrised by $q(\lambda)$.

We prove the above theorem in Appendix B.

8 Building Hardness-Preserving Distributed Samplers

We explain the idea behind our result. Consider a PPT adversary $\mathcal{A}$ that outputs 1 with non-negligible probability $\epsilon(\lambda)$ in the real-world execution of the regular and programmable lossy distributed sampler (see Fig. 20). In such execution, the distributed sampler will be in standard mode. We recall that our goal is to show the existence of a simulator, which depends on $\mathcal{A}$, such that, even in the simulated execution, the adversary $\mathcal{A}$ still outputs 1 with non-negligible probability.

We use a hybrid argument. In the first stage, we switch our distributed sampler to lossy mode. The new setting is clearly distinguishable from the initial one but, by choosing the polynomial q parametrising the lossy mode properly, we can make sure that the adversary $\mathcal{A}$ still outputs 1 with probability at least $\epsilon(\lambda)/2$.

In the next hybrid, we use the regularity of the lossy distributed sampler to argue that the probability that $\mathcal{A}$ outputs 1 and $\mathcal{Z}$ guesses the output chosen by the adversary is also non-negligible.

In the final hybrid, we rely on the programmability properties to hide an ideal sample R in the position guessed by $\mathcal{Z}$. Since the adversary cannot detect any change, $\mathcal{A}$ will still have a non-negligible probability of outputting 1 while picking R as output of the protocol.

From the last hybrid, we can easily obtain the simulators we are looking for. We simulate the CRS using $\mathsf{LossySetup}\big(\mathbb{1}^\lambda, q(\lambda)\big)$. The choice of $q(\lambda)$ depends on $\mathcal{A}$. In particular, $q(\lambda)$ needs to be sufficiently large so that $\mathcal{A}$ cannot distinguish between the first two hybrids with advantage greater than $\epsilon(\lambda)/2$. The simulation of the distributed sampler message is instead performed using $\mathsf{ProgGen}$. The programmed position is sampled using $\mathcal{Z}$. We formalise the construction in Fig. 39.

```
THE HARDNESS-PRESERVING SIMULATORS.
Let $q(\lambda)$ be the polynomial associated with $\mathcal{A}$.
$\mathsf{SimSetup}_\mathcal{A}\big(\mathbb{1}^\lambda\big)$:

  1. $(\mathsf{crs}, \zeta) \xleftarrow{\$} \mathsf{LossySetup}(\mathbb{1}^\lambda, q)$
  2. Output $\mathsf{crs}$ and $\zeta$.

$\mathsf{SimGen}_\mathcal{A}\big(\mathbb{1}^\lambda, \mathsf{sid}, i, \zeta, R\big)$:

  1. $z \xleftarrow{\$} \mathcal{Z}(\zeta)$
  2. $(U_i, \xi) \xleftarrow{\$} \mathsf{ProgGen}(\mathbb{1}^\lambda, \mathsf{sid}, i, z, R, \zeta)$
  3. Output $U_i$.
```

Fig. 39. The hardness-preserving simulators.

Theorem 17 (Hardness-preserving distributed sampler). *Let* $\mathsf{DS} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Sample}, \mathsf{SimSetup}_\mathcal{A}, \mathsf{SimGen}_\mathcal{A})$ *be a regular and programmable lossy distributed sampler for* $\mathcal{D}(\mathbb{1}^\lambda)$ *against* AClass. *Then, the construction described in Fig. 34 and Fig. 39 is an n-party hardness-preserving distributed sampler for* $\mathcal{D}$ *against* AClass.

We prove Theorem 17 in Appendix C.

8.1 Building Indistinguishability Preserving Distributed Samplers

We now explain why the distributed sampler presented in Section 7 is indistinguishability preserving.

Consider any pair of chosen-sample indistinguishable games $\mathcal{G}_0$ and $\mathcal{G}_1$ where $\mathcal{G}_0 = (\mathcal{D}, \mathsf{Ch}_0)$ is a game with oracle distribution and $\mathcal{G}_1 = (\mathcal{D}', \mathsf{Ch}_1)$ is a game

```
The Indistinguishability-Preserving Simulators
$\mathsf{SimSetup}(\mathbb{1}^\lambda)$:

  1. $(\sigma, \tau_s, \tau_e) \overset{\$}{\leftarrow} \mathsf{NIZK.SimSetup}(\mathbb{1}^\lambda)$
  2. $(\sigma', \tau') \overset{\$}{\leftarrow} \mathsf{NIZK'.SimSetup}(\mathbb{1}^\lambda)$
  3. $f \overset{\$}{\leftarrow} \mathsf{ELF.Gen}(M, M)$
  4. Output $\mathsf{crs} := (\sigma, \sigma')$ and $\zeta := (\sigma, \sigma', \tau_s, \tau_e, \tau', f)$

$\mathsf{SimGen}\big(\mathbb{1}^\lambda, \mathsf{sid}, i, \zeta := (\sigma, \sigma', \tau_s, \tau_e, \tau', f), \mathsf{aux}\big)$:

  1. $K \overset{\$}{\leftarrow} F.\mathsf{Gen}(\mathbb{1}^\lambda)$
  2. $K_2^{(i)} \overset{\$}{\leftarrow} F_2.\mathsf{Gen}(\mathbb{1}^\lambda)$
  3. $\mathsf{hk}_i \overset{\$}{\leftarrow} \mathsf{Hash.Gen}(\mathbb{1}^\lambda)$
  4. $\mathsf{EP}_i \overset{\$}{\leftarrow} \mathsf{iO}(\mathbb{1}^\lambda, \mathsf{EProg}_{\mathsf{Ls}}[K_2^{(i)}, i])$ (see Fig. 30)
  5. $\forall j \neq i: \quad \tau_e^j \overset{\$}{\leftarrow} \mathsf{NIZK.Trap}\big(\tau_e, (\mathsf{sid}, j)\big)$
  6. $\mathsf{DP}_i \overset{\$}{\leftarrow} \mathsf{iO}(\mathbb{1}^\lambda, \mathsf{DProg}_{\mathsf{IP}}[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j \neq i}, K, f, \mathsf{aux}])$ (see Fig. 41)
  7. $\pi_i \overset{\$}{\leftarrow} \mathsf{NIZK.SimProve}\big(\tau_s, (\mathsf{sid}, i), (i, \mathsf{hk}_i, \mathsf{EP}_i)\big)$
  8. $\pi'_i \overset{\$}{\leftarrow} \mathsf{NIZK'.SimProve}\big(\tau', (i, \mathsf{sid}, \mathsf{hk}_i, \mathsf{EP}_i, \mathsf{DP}_i, \pi_i, \sigma)\big)$
  9. Output $U_i := (\mathsf{hk}_i, \mathsf{EP}_i, \mathsf{DP}_i, \pi_i, \pi'_i)$ and $\xi := (\mathsf{sid}, \sigma', \sigma, f, K, \mathsf{aux})$.

$\mathsf{Trap}\big(\xi = (\mathsf{sid}, \sigma', \sigma, f, K, \mathsf{aux}), \big(U_j = (\mathsf{hk}_j, \mathsf{EP}_j, \mathsf{DP}_j, \pi_j, \pi'_j)\big)_{j \in [n]}\big)$:

  1. $\forall j \in [n]: \quad b_j \leftarrow \mathsf{NIZK'.Verify}\big(\sigma', \pi'_j, (j, \mathsf{sid}, \mathsf{hk}_j, \mathsf{EP}_j, \mathsf{DP}_j, \pi_j, \sigma)\big)$
  2. If $\exists j \in [n]$ such that $b_j = 0$, output $(\bot, \bot)$.
  3. $z \leftarrow f\big((\mathsf{hk}_j, \mathsf{EP}_j)_{j \in [n]}\big)$
  4. $s \leftarrow F(K, z)$
  5. Output $(R, T) \leftarrow \mathcal{D}'(\mathbb{1}^\lambda, \mathsf{aux};\ s)$
```

Fig. 40. The indistinguishability-preserving simulators

with trapdoored oracle distribution satisfying trapdoor security. We start by considering any PPT adversary $\mathcal{A}$ whose goal is to distinguish between the compiled games $\mathcal{G}_0'$ and $\mathcal{G}_1'$. The proof relies on a hybrid argument beginning from $\mathcal{G}_0'$. We will explain the distributed sampler simulator for the trapdoored mode in the last hybrid.

The hybrids. In the first stage, we activate the lossy mode of the distributed sampler using some polynomial $q(\lambda)$. At this point, the output of the construction is restricted in a set of polynomial size. Notice, however, that we have given the adversary non-negligible distinguishability advantage $\epsilon_1(\lambda)$. We will argue later why this will not constitute a problem.

In the next hybrid, we proceed by switching from the challenger Ch_0 to the challenger Ch_1 without providing the latter with any trapdoor T. The modification cannot be detected by the adversary due to the chosen-sample indistinguishability between $\mathcal{G}_0$ and $\mathcal{G}_1$.

Next, using obfuscation and puncturable PRFs, we will gradually change the distribution of the outputs of the distributed sampler, switching from $\mathcal{D}$ to the trapdoored distribution $\mathcal{D}'$. The technique is similar to the one we used to prove programmability. The main difference is that we repeat the procedure many times, once for each element in the image of the ELF. Simultaneously, we will start providing Ch_1 with the trapdoors T. Specifically, there will be some hybrids in which part of the distributed sampler outputs are produced using $\mathcal{D}$ whereas the rest is generated using $\mathcal{D}'$. When the distributed sampler output chosen by the adversary is generated using $\mathcal{D}'$, we provide the corresponding trapdoor T to Ch_1 otherwise, we will not. We will be able to retrieve the trapdoors leveraging the knowledge of the ELF f and the PPRF key K hardcoded into the lossy-mode messages. The randomness fed into $\mathcal{D}'$ will indeed be $F(K, z)$ where $z = f\big((\mathsf{hk}_j, \mathsf{EP}_j)_{j\in[n]}\big)$, similarly to what happened in $\mathsf{DProg}_{\mathsf{Ls}}$ (see Fig. 31). To prove that this stage is indistinguishable from the previous one, we use a hybrid argument that is iterated over the image of the ELF. Since the latter has polynomial cardinality, we do not need to assume that $\mathcal{G}_1$ satisfies subexponential trapdoor security.

In the last stage, which will correspond to $\mathcal{G}_1'$, we switch back to a construction where the outputs have high entropy. This will be done by setting the ELF in the construction back to injective mode. The distributions of the outputs will remain as in the previous hybrid, namely, with a trapdoor embedded in them. In the process, however, we will give the adversary other non-negligible advantage $\epsilon_2(\lambda)$. Notice anyway, that this stage is independent of the polynomial $q(\lambda)$.

Why are $\mathcal{G}_0'$ and $\mathcal{G}_1'$ indistinguishable? Suppose that our adversary $\mathcal{A}$ can distinguish between the initial and the final stage with non-negligible advantage $\epsilon(\lambda)$. By choosing the polynomial $q(\lambda)$ in the lossy mode properly, we can make $\epsilon_1(\lambda)$ and $\epsilon_2(\lambda)$ arbitrarily small non-negligible functions. In particular, we can make sure that no adversary running in the same time as $\mathcal{A}$ can distinguish between $\mathcal{G}_0'$ and $\mathcal{G}_1'$ with advantage greater than $\epsilon(\lambda)/2$. In this way, we reach a contradiction.

$\mathsf{DProg}_{\mathsf{IP}}[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j \neq i}, K, f, \mathsf{aux}]$

Hard-coded. The index i of the party, the session identity sid, a PPRF key $K_2^{(i)}$, the encryption program EP_i, the hash key hk_i, the extractable NIZK CRS σ and the extraction trapdoors $(\tau_e^j)_{j \neq i}$, the PPRF key K, the ELF f, the auxiliary information aux.
Input. Set of $n-1$ tuples $(\mathsf{hk}_j, \mathsf{EP}_j, \pi_j)_{j \neq i}$.

```
$\forall j \neq i :\quad b_j \leftarrow \mathsf{NIZK.Verify}\big(\sigma, (\mathsf{sid}, j), \pi_j, (j, \mathsf{hk}_j, \mathsf{EP}_j)\big)$
$\forall j \neq i :\quad \big(K_1^{(j)}, K_2^{(j)}\big) \leftarrow \mathsf{NIZK.Extract}\big(\tau_e^j, \pi_j, (j, \mathsf{hk}_j, \mathsf{EP}_j)\big)^a$
If $\exists j \neq i$ such that $b_j = 0$ or $\big(K_1^{(j)}, K_2^{(j)}\big) = \bot$, output $\bot$
$\forall j \in [n] :\quad y_j \leftarrow \mathsf{Hash}\big(\mathsf{hk}_j, (\mathsf{hk}_l, \mathsf{EP}_l)_{l \neq j}\big)$
$\forall j \neq i :\quad s_j \leftarrow F_1\big(K_1^{(j)}, y_j\big)$
$\forall j \in [n] :\quad (r_j, r'_j, r''_j, \eta_j, \eta'_j) \leftarrow F_2\big(K_2^{(j)}, y_j\big)$
$z \leftarrow f\big((\mathsf{hk}_j, \mathsf{EP}_j)_{j \in [n]}\big)$
$s \leftarrow F(K, z)$
$(\hat{R}, \hat{T}) \leftarrow \mathcal{D}'(\mathbb{1}^\lambda, \mathsf{aux};\ s)$
$(\phi, \mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{mkFHE.Sim}_1(\mathbb{1}^\lambda, i;\ r''_i)$
$d_i \leftarrow \mathsf{mkFHE.Sim}_2\Big(\phi, \tilde{\mathcal{D}}, \hat{R}, (s_j, r_j, r'_j)_{j \neq i};\ \eta'_i\Big)$ (see bottom of Fig. 29)
Output $d_i$
```

Fig. 41. The unobfuscated decryption program for the indistinguishability-preserving simulator

The simulators. From the last stage of our hybrid argument, we can easily derive the simulators for the indistinguishability-preserving distributed sampler. The algorithm $\mathsf{SimSetup}$ will simulate the CRSs for NIZK and NIZK' as $\mathsf{LossySetup}$ did (see Fig. 36). Furthermore, it will generate an injective-mode ELF f. The simulator SimGen will behave exactly as $\mathsf{LossyGen}$ (see Fig. 36) with the exception that, in DP_i, we substitute $\mathcal{D}(\mathbb{1}^\lambda)$ with $\mathcal{D}'(\mathbb{1}^\lambda, \mathsf{aux})$. The trapdoor information ξ will contain the ELF f, the PPRF key K and aux. This information is sufficient to retrieve the trapdoors hidden in the distributed sampler outputs. We formalise the construction in Fig. 40.

Theorem 18 (Indistinguishability-preserving distributed sampler). *Let $\mathcal{D}$ be an efficient distribution and let $\mathcal{D}'$ be a trapdoored distribution for $\mathcal{D}$. Assume that* ELF *is a regular extremely lossy function. Under the hypothesis of Theorem 16, the construction* $\mathsf{DS} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Sample}, \mathsf{SimSetup}, \mathsf{SimGen}, \mathsf{Trap})$ *described in Fig. 34 and Fig. 40 is an n-party indistinguishability-preserving distributed sampler for $(\mathcal{D}, \mathcal{D}')$ against* AClass.

Observe that in the non-uniform setting, we can instantiate the construction so that the CRS is statistically close to uniform and its length depends only on the security parameter. In the uniform setting, instead, we do not need any CRS. We prove Theorem 18 in Appendix D.

9 NIZKs with no CRS in the Uniform Setting

Non-interactive zero-knowledge proofs (NIZKs) allow proving the truthfulness of a statement by sending a single message and without revealing further information. It is easy to see that NIZKs for generic NP languages always need CRSs [BP04]: zero-knowledge implies the existence of valid proofs for false statements[14]. If any construction did not rely on a CRS, non-uniform adversaries that receive accepting proofs for false statements as part of their advice string would break soundness. One could ask if CRSs are still needed if we aim for security against uniform adversaries. Barak and Pass [BP04] showed that this is not the case: in their work, they presented NIZKs without CRS achieving soundness and zero-knowledge against uniform adversaries. In this section, in contrast with the approach of [BP04], we study the problem of CRS-less NIZKs in a concurrent setting, explaining how to achieve, among other notions, multi-theorem zero-knowledge, simulation-soundness and simulation extractability. Throughout the paper, we use the term U-NIZK to denote non-interactive proofs achieving security against uniform adversaries.

We start by presenting definitions. Although our ultimate goal is to remove CRSs, for generality and to keep the notation consistent with the non-uniform setting, our notions will still rely on a Setup algorithm. In the CRS-less constructions, Setup will output the empty string.

Multi-theorem zero-knowledge U-NIZK. We begin by formalising the notion of multi-theorem zero-knowledge against uniform adversaries. The concept is analogous to the non-uniform setting: the adversary cannot distinguish between an oracle that generates proofs using the witnesses and one that simulates them.

In order to achieve a weak form of composability for U-NIZKs, we also provide the adversary with a non-uniform advice string a_λ. While in the non-uniform setting, we ask zero-knowledge to hold independently of the choice of the advice string, we now define security with respect to a fixed sequence of advice strings $a = (a_\lambda)_{\lambda\in\mathbb{N}}$. We will use this approach also in all the other definitions of this section.

Definition 33 (Multi-Theorem Zero-Knowledge). *Let $a = (a_\lambda)_{\lambda\in\mathbb{N}}$ be non-uniform advice. A NIZK* $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ *for the NP relation $\mathcal{R}$ is a-disclosed multi-theorem zero-knowledge if there exists a possibly non-uniform PPT algorithm* $\mathsf{SimSetup}$ *and a uniform PPT algorithm* $\mathsf{SimProve}$ *such that no uniform PPT adversary can win the game in Fig. 42 with non-negligible advantage.*

Simulation-Sound U-NIZKs. We now present the definition of simulation-sound U-NIZK. This is a multi-theorem zero-knowledge construction in which it is hard

[14] Consider e.g. the language consisting of the image of a PRG. Since it is hard to distinguish between true statements and false statements, by running the zero-knowledge simulator on a random false statement, we obtain an accepting NIZK with high probability.

Multi-Theorem Zero-Knowledge Game for U-NIZKs

Initialisation: This procedure is run only once, at the beginning of the game.

1. $b \xleftarrow{\$} \{0,1\}$
2. $\sigma_0 \xleftarrow{\$} \mathsf{Setup}(\mathbb{1}^\lambda)$
3. $(\sigma_1, \tau) \xleftarrow{\$} \mathsf{SimSetup}(\mathbb{1}^\lambda)$
4. Activate the adversary with $\mathbb{1}^\lambda$, σ_b and a_λ.

Prove: This procedure can be queried multiple times. Upon receiving any query (Prove, x, w) where $(x, w) \in \mathcal{R}$, compute the following.

1. $\pi^0 \xleftarrow{\$} \mathsf{Prove}(\mathbb{1}^\lambda, \sigma, x, w)$
2. $\pi^1 \xleftarrow{\$} \mathsf{SimProve}(\tau, x)$
3. Give π^b to the adversary.

Win: The adversary wins if it guesses b.

Fig. 42. Multi-theorem zero-knowledge game for U-NIZKs

to generate accepting proofs for false statements even when the adversary has access to the simulation oracle.

Definition 34 (Simulation-sound U-NIZK). *Let $a = (a_\lambda)_{\lambda \in \mathbb{N}}$ be non-uniform advice. A NIZK* $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ *for $\mathcal{R}$ is a-disclosed U-simulation-sound if it is a-disclosed multi-theorem zero-knowledge and, for every uniform PPT adversary $\mathcal{A}$,*

$$\Pr\left[\begin{array}{c} (\pi, x) \notin Q \\ \mathsf{Verify}(\sigma, \pi, x) = 1 \\ x \notin L_{\mathcal{R}} \end{array} \middle| \begin{array}{l} (\sigma, \tau) \xleftarrow{\$} \mathsf{SimSetup}(\mathbb{1}^\lambda) \\ (\pi, x) \xleftarrow{\$} \mathcal{A}^{\mathsf{SimProve}(\tau, \cdot)}(\mathbb{1}^\lambda, \sigma, a_\lambda) \end{array}\right] = \mathsf{negl}(\lambda)$$

where Q denotes the set of the responses provided by $\mathsf{SimProve}(\tau, \cdot)$.

Restricted simulation-sound U-NIZK. Restricted simulation-soundness is a weaker form of simulation-soundness: in the corresponding game, the adversary can only query the simulation oracle with statements for which it knows a witness. The primitive guarantees the existence of an inefficient uniform algorithm Check that rejects all proofs for false statements, even those that pass the usual verification. The primitive guarantees the hardness of generating proofs for which Check and Verify disagree. Observe that there might exist proofs for true statements that are rejected by Check.

While building CRS-less simulation-sound U-NIZKs will require introducing new assumptions, we achieved restricted simulation-soundness from more conservative hypotheses.

Definition 35 (Restricted simulation-sound U-NIZK). *Let $a = (a_\lambda)_{\lambda \in \mathbb{N}}$ be non-uniform advice. A NIZK* $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ *for $\mathcal{R}$ is a-disclosed re-*

Game for Restricted Simulation-Sound U-NIZKs

Initialisation: This procedure is run only once, at the beginning of the game.

1. $Q \leftarrow \emptyset$
2. $(\sigma, \tau) \xleftarrow{\$} \mathsf{SimSetup}(\mathbb{1}^\lambda)$
3. Activate the adversary with $\mathbb{1}^\lambda$, σ and a_λ.

Prove: This procedure can be queried multiple times. Upon receiving any query (Prove, x, w) where $(x, w) \in \mathcal{R}$, compute the following.

1. $\pi \xleftarrow{\$} \mathsf{SimProve}(\tau, x)$
2. Add (π, x) to Q
3. Give π to the adversary.

Win: The adversary wins if it outputs $(\pi, x) \notin Q$ such that $\mathsf{Verify}(\sigma, \pi, x) \neq \mathsf{Check}(\sigma, \pi, x)$.

Fig. 43. Restricted simulation-sound U-NIZK game

stricted U-simulation-sound if it is a-disclosed multi-theorem zero-knowledge and there exists an inefficient, uniform, deterministic algorithm Check *such that*

1. *For every pair* (σ, π, x) *such that* $\mathsf{Verify}(\sigma, \pi, x) = 1$ *but* $x \notin L_{\mathcal{R}}$
$$\Pr\Big[\mathsf{Check}(\sigma, \pi, x) = 1\Big] = 0.$$
2. *No uniform PPT adversary* $\mathcal{A}$ *can win the game in Fig. 43 with non-negligible probability.*

Observe that Check can trivially be the algorithm that performs a brute-force search for a witness, however, in this case, the construction would have limited applications. For instance, consider the case in which n parties, some of them corrupted, exchange messages along with NIZKs proving their well-formedness. Let $S(\lambda)$ denote the running time of Check. Using a restricted simulation-sound NIZK, we can switch to the hybrid in which the honest parties send simulated proofs and abort if the proof of any corrupted player is rejected by Check. Notice that at this point, the simulation is inefficient as it needs to run Check. Now, suppose that we want to substitute the messages of the honest parties with a new set of indistinguishable messages not belonging to the language. The operation can be performed as long as distinguishing is hard even for adversaries running in $\mathsf{poly}\big(\lambda, S(\lambda)\big)$ time. If, however, Check performs a brute-force search for the witness, that can never happen.

Simulation-extractable U-NIZK. A simulation-extractable U-NIZK satisfies knowledge soundness: the trapdoor τ generated by the setup simulator allows us to efficiently extract the witness from verifying proofs. Security states that even if the adversaries has access to the simulation oracle, it is hard to generate a proof that verifies but for which the extraction of the witness fails.

Definition 36 (Simulation-extractable U-NIZK). *Let $a = (a_\lambda)_{\lambda \in \mathbb{N}}$ be non-uniform advice. A NIZK* ($\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify}$) *for $\mathcal{R}$ is a-disclosed U-simulation-extractable if it is a-disclosed multi-theorem zero-knowledge and there exists a uniform PPT algorithm* $\mathsf{Extract}$ *such that*

1. *The algorithm* $\mathsf{Extract}$ *is deterministic and, for every $w = \mathsf{Extract}(\tau, \pi, x)$,*

$$\Pr\Big[(x,w) \in \mathcal{R} \Big| w \neq \bot\Big] = 1.$$

2. *For every uniform PPT adversary $\mathcal{A}$,*

$$\Pr\left[\begin{array}{c} (\pi, x) \notin Q \\ \mathsf{Verify}(\sigma, \pi, x) = 1 \\ \mathsf{Extract}(\tau, \pi, x) = \bot \end{array} \middle| \begin{array}{l} (\sigma, \tau) \xleftarrow{\$} \mathsf{SimSetup}(\mathbb{1}^\lambda) \\ (\pi, x) \xleftarrow{\$} \mathcal{A}^{\mathsf{SimProve}(\tau, \cdot)}(\mathbb{1}^\lambda, \sigma, a_\lambda) \end{array}\right] = \mathsf{negl}(\lambda)$$

where Q denotes the set of the responses provided by $\mathsf{SimProve}(\tau, \cdot)$.

Restricted simulation-extractable U-NIZK. Similarly to simulation-soundness, we define a weaker notion of simulation-extractability by restricting the simulation queries to statements for which the adversary knows a witness. Compared to simulation-extractability, we will be able to realise this definition without CRSs using weaker assumptions.

Definition 37 (Restricted simulation-extractable U-NIZK). *Let $a = (a_\lambda)_{\lambda \in \mathbb{N}}$ be non-uniform advice. A NIZK* ($\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify}$) *for $\mathcal{R}$ is a-disclosed restricted U-simulation-extractable if it is a-disclosed multi-theorem zero-knowledge and there exists a uniform PPT algorithm* $\mathsf{Extract}$ *such that*

1. *The algorithm* $\mathsf{Extract}$ *is deterministic and, for every $w = \mathsf{Extract}(\tau, \pi, x)$,*

$$\Pr\Big[(x,w) \in \mathcal{R} \Big| w \neq \bot\Big] = 1.$$

2. *No uniform PPT adversary $\mathcal{A}$ can win the game in Fig. 44 with non-negligible probability.*

Computable sequence and deterministic U-NIZKs. Before describing our CRS-less constructions, we present two additional definitions. These will be fundamental to formalise a weak form of composability for U-NIZKs.

The first definition introduces the notion of $T(\lambda)$-computable sequence and quantumly computable sequence. The former consists of a sequence of values indexed by the security parameter that is computable in uniform-$\mathsf{poly}\big(\lambda, T(\lambda)\big)$ time. The latter indicates a sequence that is computable in uniform, quantum polynomial time.

Game for Restricted Simulation-Extractable U-NIZKs

Initialisation: This procedure is run only once, at the beginning of the game.

1. $Q \leftarrow \emptyset$
2. $(\sigma, \tau) \xleftarrow{\$} \mathsf{SimSetup}(\mathbb{1}^\lambda)$
3. Activate the adversary with $\mathbb{1}^\lambda$, σ and a_λ.

Prove: This procedure can be queried multiple times. Upon receiving any query (Prove, x, w) where $(x, w) \in \mathcal{R}$, compute the following.

1. $\pi \xleftarrow{\$} \mathsf{SimProve}(\tau, x)$
2. Add (π, x) to Q
3. Give π to the adversary.

Win: The adversary wins if it outputs a pair $(\pi, x) \notin Q$ such that $\mathsf{Verify}(\sigma, \pi, x) = 1$ but $\mathsf{Extract}(\tau, \pi, x) = \bot$.

Fig. 44. Restricted simulation-extractable U-NIZK game

Definition 38 (T-computable sequence and quantumly computable sequence). *Let $a := (a_\lambda)_{\lambda \in \mathbb{N}}$ be a sequence of values. Let $T(\lambda)$ be a function of λ. We say that a is T-computable if there exists a uniform algorithm $\mathcal{C}$ running in $\mathsf{poly}\big(\lambda, T(\lambda)\big)$ time such that*

$$\Pr\Big[\mathcal{C}(\mathbb{1}^\lambda, \mathbb{1}^{T(\lambda)}) = a_\lambda\Big] = 1 - \mathsf{negl}(\lambda).$$

We say that a is quantumly computable if there exists a uniform, quantum polynomial time algorithm $\mathcal{Q}$ such that

$$\Pr\Big[\mathcal{Q}(\mathbb{1}^\lambda) = a_\lambda\Big] = 1 - \mathsf{negl}(\lambda).$$

Our second definition formalises the concept of $T(\lambda)$-deterministic U-NIZK, i.e. a U-NIZK in which $\mathsf{SimSetup}$ is deterministic and its output is $T(\lambda)$-computable.

Definition 39 (S-deterministic U-NIZK). *Let $T(\lambda)$ be a function of λ. We say that a U-NIZK is T-deterministic if there exists an T-computable sequence $(a_\lambda)_{\lambda \in \mathbb{N}}$ such that*

$$\Pr\Big[\mathsf{SimSetup}(\mathbb{1}^\lambda) = a_\lambda\Big] = 1.$$

9.1 Building restricted simulation-sound U-NIZKs without CRS

In this section, we show how to build restricted simulation-sound U-NIZK without CRS. In [BL18b], Bitansky and Lin presented CRS-less NIZKs with a weak form of security against non-uniform adversaries: an attacker can generate valid proofs only for a number of false statements proportional to its non-uniformity.

It is not hard to see that their constructions achieve full soundness against uniform adversaries (using collision resistant keyless hash functions as incompressible problem). One of them is particularly interesting: it satisfies a tag-based version of restricted simulation-soundness[15] (i.e. each proof is associated with a tag. Generating a valid proof for a false statement under a tag $\widehat{\mathsf{tag}}$ remains hard even given simulated proofs under tags different from $\widehat{\mathsf{tag}}$).

In this section, we show how to achieve restricted simulation-soundness without needing tags. Following the blueprint of [BL18b], our construction is obtained by using tag-based non-malleable commitments in the scheme of [BP04]. In particular, each proof includes a CCA commitment to 0, we denote it by c. We use a NIWI to prove that either our statement lies in the language or c is a commitment to a collision for a keyless hash function. Following [GO07], the commitment tag will consist of a strong one-time signature verification key vk. The proof is finally augmented with a signature over the NIWI and the commitment using vk. The scheme is formalised in Fig. 45.

To simulate a proof, we will commit to a collision for the keyless hash function and use the latter as witness for the NIWI. It is easy to prove zero-knowledge by leveraging witness indistinguishability and the hiding properties of the commitment.

As for soundness, thanks to the collision resistance of the hash function and the perfect soundness of the NIWI, it will be hard to generate a proof for a false statement. Indeed, in all such proofs, c is a commitment to a collision. The signature and the CCA security of the commitment will make sure that this holds even if the adversary has access to an oracle that generates simulated proofs. Indeed, the signature guarantees that, if the adversary succeeds in generating a valid proof for a false statement, it must use a fresh signing key, instead of one generated by the oracle. Since the tag of the CCA commitment is the verification key, in order to obtain a commitment to a collision under its own tag, the adversary needs to either break hiding or the non-malleability of the commitment. In either case, that would contradict the security of the CCA commitment.

Observe that Check can easily reject all proofs for false statements by extracting the value in c and checking if it consists of a collision for the hash function. The running time of Check is therefore independent of $\mathcal{R}$.

Formalising the construction. Let $T(\lambda)$ be a function of the security parameter and let $e \in \mathbb{N}$ such that $T(\lambda) \leq 2^{\lambda^e}$. We rely on a e-computation enabled non-interactive CCA commitment [GKLW21]. We require that the construction satisfies perfect correctness. Let $S(\lambda)$ denote the running time of $\mathsf{CCACom.Val}$.

Let KHash be a keyless collision-resistant hash function with domain size $p(\lambda)$ achieving security against uniform adversaries running in $\mathsf{poly}\big(\lambda, S(\lambda), T(\lambda)\big)$

[15] In the paper [BL18b], Bitansky and Lin claim that the construction is tag-based simulation-sound. This is, however, not the case: the construction is only restricted simulation-sound as the NIWI in their scheme guarantees that the trapdoors are kept secret only if the statement lies in the language.

time. Suppose also that the smallest collision for KHash according to the lexicographical order is 2^{λ^e}-computable. Fo every $\lambda \in \mathbb{N}$, we denote this collision by $(y^0_\lambda, y^1_\lambda)$.

Let NIWI be a perfectly sound, witness indistinguishable proof system for the relation

$$\mathcal{R}_{\mathsf{NIWI}} := \left\{ \begin{matrix} (\mathsf{vk}, c, x), \\ w \end{matrix} \middle| (x, w) \in \mathcal{R} \quad \text{OR} \quad \begin{pmatrix} w = (y_0, y_1, r) \\ y_0 \neq y_1 \\ \mathsf{KHash}(y_0) = \mathsf{KHash}(y_1) \\ c = \mathsf{CCACom}\big(\mathbb{1}^\lambda, \mathsf{vk}, (y_0, y_1);\ r\big) \end{pmatrix} \right\}$$

We require that NIWI is witness indistinguishable even against adversaries running in $\mathsf{poly}\big(\lambda, S(\lambda)\big)$ time. Finally, let $\mathsf{SOTS} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ be a strong one-time signature. To summarise, we have $T(\lambda), S(\lambda) \ll 2^{\lambda^e}$.

Theorem 19. *Let $a := (a_\lambda)_{\lambda\in\mathbb{N}}$ be a T-computable sequence. Assume the existence of computation-enabled CCA commitments with $\mathsf{poly}(\lambda)$ tag size and strong one-time signatures. Furthermore, assume the existence of subexponentially collision-resistant keyless hash functions and subexponentially witness-indistinguishable non-interactive proofs without CRS. Then, the construction in Fig. 45 is an a-disclosed restricted simulation-sound NIZK without CRS for $\mathcal{R}$ with security against uniform PPT adversaries.*

Proof. Completeness follows immediately from the completeness of NIWI.

The first property of restricted simulation-soundness is an easy consequence of the perfect soundness of NIWI and the perfect correctness of CCACom. Indeed, if the NIZK verifies, the NIWI π' must verify. If x does not belong to the language, it must be that c is a commitment to a collision for KHash. So Check outputs 0 with probability 1.

We now prove the second property and multi-theorem zero-knowledge at once. In particular, consider the zero-knowledge game in Fig. 42 in which we augment the adversary with a verification oracle. The latter can be queried with pairs (π, x) that are not responses of the proving oracle. When $b = 0$, the answer will be $\mathsf{Verify}(\sigma, \pi, x)$, when $b = 1$ instead, the answer will be $\mathsf{Check}(\sigma, \pi, x)$. We show that even in this game, the adversary cannot guess b with non-negligible advantage. This immediately implies both multi-theorem zero-knowledge and restricted simulation-soundness. Indeed, if there was a way to generate a pair (π, x) such that $\mathsf{Verify}(\sigma, \pi, x) \neq \mathsf{Check}(\sigma, \pi, x)$ while having access to the simulation oracle, then it is immediate to guess b: when $b = 0$, the verification oracle always outputs $\mathsf{Verify}(\sigma, \pi, x)$, when $b = 1$, instead, that does not happen.

We use a hybrid argument.

Hybrid 0. This corresponds to the augmented zero-knowledge game when $b = 0$. In particular, the proofs are generated using witnesses and verification just relies on Verify.

Hybrid 1. In this hybrid, in each verification query where $(\pi, x) \notin Q$, the challenger checks if the verification key vk included in π coincides with one stored in Q. In that case, the challenger immediately returns 0.

A RESTRICTED SIMULATION-SOUND NIZK WITHOUT CRS

```
$\mathsf{Prove}(1\!\!1^\lambda, x, w)$

  1. $(\mathsf{vk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{SOTS.Gen}(1\!\!1^\lambda)$
  2. $c \xleftarrow{\$} \mathsf{CCACom}(1\!\!1^\lambda, \mathsf{vk}, 0^{2p(\lambda)})$
  3. $\pi' \xleftarrow{\$} \mathsf{NIWI.Prove}\big(1\!\!1^\lambda, (\mathsf{vk}, c, x), w\big)$
  4. $s \leftarrow \mathsf{SOTS.Sign}\big(\mathsf{sk}, (\mathsf{vk}, c, \pi')\big)$
  5. Output $\pi := (\mathsf{vk}, c, \pi', s)$

$\mathsf{Verify}\big(\pi = (\mathsf{vk}, c, \pi', s), x\big)$

  1. $b_0 \leftarrow \mathsf{SOTS.Verify}\big(\mathsf{vk}, (\mathsf{vk}, c, \pi'), s\big)$
  2. $b_1 \leftarrow \mathsf{NIWI.Verify}\big(\pi', (\mathsf{vk}, c, x)\big)$
  3. Output $b_0 \wedge b_1$.

$\mathsf{SimSetup}(1\!\!1^\lambda)$

  1. Retrieve the smallest collision $(y_\lambda^0, y_\lambda^1)$ for $\mathsf{KHash}$ according to the lexicographical order. Such collision is given as a non-uniform advice.
  2. Output the empty string along with $\tau = (y_\lambda^0, y_\lambda^1)$.

$\mathsf{SimProve}(\tau = (y_\lambda^0, y_\lambda^1), x)$

  1. $(\mathsf{vk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{SOTS.Gen}(1\!\!1^\lambda)$
  2. $r \xleftarrow{\$} \{0,1\}^{q(\lambda)}$
  3. $c \leftarrow \mathsf{CCACom}(1\!\!1^\lambda, \mathsf{vk}, \tau;\, r)$
  4. $\pi' \xleftarrow{\$} \mathsf{NIWI.Prove}\big(1\!\!1^\lambda, (\mathsf{vk}, c, x), (\tau, r)\big)$
  5. $s \leftarrow \mathsf{SOTS.Sign}\big(\mathsf{sk}, (\mathsf{vk}, c, \pi')\big)$
  6. Output $\pi := (\mathsf{vk}, c, \pi', s)$

$\mathsf{Check}(\pi = (\mathsf{vk}, c, \pi', s), x)$

  1. $b \leftarrow \mathsf{Verify}(1\!\!1^\lambda, \pi, x)$
  2. If $b = 0$, output 0.
  3. $(y_0, y_1) \leftarrow \mathsf{CCACom.Val}(\mathsf{vk}, c)$
  4. If $\mathsf{KHash}(y_0) = \mathsf{KHash}(y_1)$ and $y_0 \neq y_1$, output 0, otherwise, output 1.
```

Fig. 45. A restricted simulation-sound NIZK without CRS

Claim. No PPT adversary can distinguish between Hybrid 0 and Hybrid 1.

Proof of the claim. If an adversary $\mathcal{A}$ distinguishes between Hybrid 0 and Hybrid 1, it means that, with non-negligible probability $\epsilon(\lambda)$, it can generate a proof $\pi = (\mathsf{vk}, c, \pi', s)$ for a statement x such that $\mathsf{Verify}(\pi, x) = 1$ and vk coincides with the verification key of a proof $\widehat{\pi} = (\mathsf{vk}, \widehat{c}, \widehat{\pi}', \widehat{s})$ previously generated by the challenger. Observe that $\pi \neq \widehat{\pi}$ and $\mathsf{SOTS.Verify}\big(\mathsf{vk}, (\mathsf{vk}, c, \pi'), s\big) = 1$.

We show how to build a uniform adversary $\mathcal{B}$ that breaks the security of the strong one-time signature scheme SOTS. This immediately leads to a contradiction.

Let $M := M(\lambda)$ denote a polynomial upper-bound on the number of Prove queries issued by $\mathcal{A}$. The adversary $\mathcal{B}$ receives the verification key $\widehat{\mathsf{vk}}$ from its challenger, samples a random $i \xleftarrow{\$} [M]$ and simulates the game in Hybrid 0 for an internal copy of $\mathcal{A}$. Since the one-time signature is secure even against non-uniform adversaries, we can assume that $\mathcal{B}$ receives a_λ as part of its non-uniform advice. At the i-th Prove query, $\mathcal{B}$ deviates from the game: it generates the commitment $\widehat{c}$ and the NIWI proof $\widehat{\pi}'$ using the verification key $\widehat{\mathsf{vk}}$. Then, it queries $(\widehat{\mathsf{vk}}, \widehat{c}, \widehat{\pi}')$ and includes the answer $\widehat{s}$ in the proof $(\widehat{\mathsf{vk}}, \widehat{c}, \widehat{\pi}', \widehat{s})$.

For every valid verification query $(\pi, x) \notin Q$ where $\pi = (\mathsf{vk}, c, \pi', s)$, $\mathcal{B}$ checks if the verification key in π coincides with $\widehat{\mathsf{vk}}$. If that happens and $\mathsf{Verify}(\pi, x) = 1$, $\mathcal{B}$ outputs the forgery $(\mathsf{vk}, c, \pi'), s$. Notice that $\mathcal{B}$ succeeds with probability at least $\epsilon(\lambda)/M(\lambda)$. ■

Hybrid 2. In this hybrid, we answer every verification query using Check instead of Verify. The rest remains as in Hybrid 1. In particular, we keep checking if the signature keys included in the verification queries coincide with those in Q. When that happens, we always answer with 0.

Claim. No uniform PPT adversary can distinguish between Hybrid 2 and Hybrid 1.

Proof of the claim. If an adversary $\mathcal{A}$ distinguishes between Hybrid 1 and Hybrid 2, it means that, with non-negligible probability, it can generate a proof $\pi = (\mathsf{vk}, c, \pi', s)$ for a statement x such that $\mathsf{Verify}(\pi, x) = 1$ but $\mathsf{Check}(\pi, x) = 0$. In other words, $\mathsf{CCACom.Val}(\mathsf{vk}, c)$ will output a collision for KHash.

We show how to build a uniform adversary $\mathcal{B}$ that finds a collision for KHash running in $\mathsf{poly}\big(\lambda, S(\lambda)\big)$ time. This immediately proves our claim as it contradicts the security of the keyless hash function.

The adversary $\mathcal{B}$ simulates the game in Hybrid 1 for an internal copy of $\mathcal{A}$. It retrieves a_λ in uniform $\mathsf{poly}\big(\lambda, T(\lambda)\big)$ time. For every verification query, $\mathcal{B}$ runs $\mathsf{CCACom.Val}$ on the provided commitment. Notice that this operation can be performed in uniform $\mathsf{poly}\big(\lambda, S(\lambda)\big)$ time. In case it obtains a collision for KHash, $\mathcal{B}$ outputs it. ■

Hybrid 3. This hybrid, the challenger generates the answers to Prove queries using $\mathsf{SimProve}$. All the rest remains as in Hybrid 2.

Claim. No uniform PPT adversary can distinguish between Hybrid 2 and Hybrid 3.

Proof of the claim. Let $M(\lambda)$ be a polynomial upper-bounding the number of tuples (x_i, w_i) queried by $\mathcal{A}$. Since $\mathcal{A}$ is PPT, we know that M exists. Let π_i denote the answer to the i-th query.

For every $i \in [M] \cup \{0\}$, we define Hybrid 2.i as the hybrid in which, for every $j \leq i$, we generate π_j using $\mathsf{SimProve}(\tau, x_j)$. For very $j > i$ instead, we generate π_j using $\mathsf{Prove}(\mathbb{1}^\lambda, x_j, w_j)$. The answer to the verification queries remains as in Hybrid 2.

Notice that Hybrid 2.0 is identical to Hybrid 2. Hybrid 2.M is instead identical to Hybrid 3. In order to prove our claim, it is sufficient to prove that no uniform PPT adversary can distinguish between Hybrid 2.$(i-1)$ and Hybrid 2.i for a randomly sampled $i \xleftarrow{\$} [M]$.

We show this by relying on a sequence of indistinguishable subhybrids.

Hybrid' 0. This hybrid coincides with Hybrid 2.$(i-1)$ for $i \xleftarrow{\$} [M]$.

Hybrid' 1. In this hybrid, we change the proof π_i. In particular, instead of committing to $0^{2p(\lambda)}$, we commit to the smallest collision $(y^0_\lambda, y^1_\lambda)$ for KHash according to the lexicographical order. All the rest remains as in Hybrid' 0.

This hybrid is indistinguishable from the previous one by the e-computation enabled CCA hiding property of CCACom. In the reduction, the adversary $\mathcal{B}$ starts its execution by querying the Turing machine P that computes $\tau = (y^0_\lambda, y^1_\lambda)$ and a_λ. Then, it sample $i \xleftarrow{\$} [M]$ and runs $\mathcal{A}$. It replies to the first $i-1$ queries using $\mathsf{SimProve}$. Starting from the $(i+1)$-th query, it instead uses Prove.

The adversary $\mathcal{B}$ deviates from the game in the i-th Prove query. It begins by generating the signature key pair $(\widehat{\mathsf{vk}}, \widehat{\mathsf{sk}})$ and queries $(\mathsf{tag} = \widehat{\mathsf{vk}}, 0^{2p(\lambda)}, \tau)$ to its challenger. It then uses the answer c and $(\widehat{\mathsf{vk}}, \widehat{\mathsf{sk}})$ to generate π_i. The verification queries are answered as in Hybrid 2. Notice that $\mathcal{B}$ does not need to run $\mathsf{CCACom.Val}$ as its challenger gives oracle access to it. Observe also that, with overwhelming probability, $\mathcal{B}$ does not query $(\widehat{\mathsf{vk}}, c')$ to the $\mathsf{CCACom.Val}$ oracle for any value c'. Indeed, the probability that $\widehat{\mathsf{vk}}$ was included in a verification query issued before the i-th Prove query is negligible. Furthermore, if the adversary ever uses $\widehat{\mathsf{vk}}$ in a verification query after the i-th Prove query, $\mathcal{B}$ can immediately output 0.

The adversary $\mathcal{B}$ terminates the execution outputting the same value as $\mathcal{A}$. Notice that if $\mathcal{A}$ distinguishes between Hybrid' 0 and Hybrid' 1, then $\mathcal{B}$ is a e-computation enabled adversary breaking the hiding properties of CCACom.

Hybrid' 2. In this hybrid, we change again the proof π_i. In particular, instead of using w_i as witness for NIWI, we use τ and the randomness used to commit to it. This hybrid is indistinguishable from the previous one by the $S(\lambda)$-witness indistinguishability of NIWI.

Notice that NIWI is secure against non-uniform adversaries so, in the reduction, we can assume that the adversary is given $\tau = (y^0_\lambda, y^1_\lambda)$ and a_λ as part of its advice string. The adversary we construct, denoted by $\mathcal{B}$, runs in $\mathsf{poly}\big(\lambda, S(\lambda)\big)$ time. It starts its execution sampling a random $i \xleftarrow{\$} [M]$ and it simulates the game as in Hybrid' 1 to an internal copy of $\mathcal{A}$. It deviates from the game at the i-th Prove query. In particular, after generating vk and c as usual, it queries (vk, c, x_i) along with the witnesses w_i and (τ, r) to the NIWI challenger. Here, r denotes the randomness used for the generation of c. The answer π' is included in the proof returned to $\mathcal{A}$. The adversary $\mathcal{B}$ replies to the verification queries as in Hybrid' 1. Notice that the evaluations of $\mathsf{CCACom.Val}$ take $S(\lambda)$ time. At the end of its execution, $\mathcal{B}$ outputs the same bit as $\mathcal{A}$. Notice that if $\mathcal{A}$ succeeds in distinguishing, $\mathcal{B}$ succeeds too.

Observe that Hybrid' 2 is identical to Hybrid 2.i for a random $i \xleftarrow{\$} [M]$. This ends the proof of the claim. ■

Hybrid 4. This hybrids corresponds to the augmented zero-knowledge game when $b = 1$. In particular, in the verification queries, we do not check anymore if vk is part of any element in Q, we simply run Check.

It is easy to see that Hybrid 3 and Hybrid 4 are indistinguishable. Indeed, if that was not the case, it means that, in Hybrid 3, with non-negligible probability, the adversary could generate a proof $\pi = (\mathsf{vk}, c, \pi', s)$ and a statement x such that $(\pi, x) \notin Q$, $\mathsf{Check}(\pi, x) = 1$ and vk was already used by a proof in Q. This contradicts the fact that $\mathcal{A}$ cannot distinguish between Hybrid 0 and Hybrid 3. □

9.2 Building simulation-sound U-NIZKs without CRS

In this section, we show how to build simulation-sound U-NIZKs without CRS. We present two types of constructions. In both cases, in order to prove security, we introduce new assumptions.

Uniform-DDH, uniform-LWE and challengeless one-way functions. The first scheme relies on new variants of DDH and LWE for the uniform setting. For both assumptions, the approach is the same: we require the existence of a uniform *deterministic* algorithm that generates part of the DDH and LWE challenge. Specifically, for DDH, the algorithm outputs the group and the first two elements of the DDH tuple. For LWE, the algorithm outputs the matrix A describing the lattice. One can imagine these algorithms as the result of generating two group elements g and h (in the case of DDH), or a full-rank matrix A (in the case of LWE) using a SHA hash function. The idea is that, if the adversary is uniform, deriving $a \in \mathbb{Z}$ such that $g^a = h$ or a small vector $u \neq 0$ such that $A \cdot u = 0$ is hard. Notice that the assumption totally breaks if we consider non-uniform adversaries as they can receive a and u as part of their non-uniform advice. In the case of DDH, our assumption is that no uniform adversary can distinguish between the pair (g^r, h^r) and (g^r, g^s) where r and s are random. In the LWE case, instead, we assume that no uniform adversary can distinguish between a uniformly random vector v and $A^\intercal \cdot s + x$, where s is uniformly random and x is a random small vector. We further assume that both assumptions are subexponentially secure against uniform adversaries and, in the case of LWE, even against uniform quantum adversaries.

Definition 40 (Uniform DDH). *Let* DDHGen *be a uniform, polynomial time deterministic algorithm that on input the security parameter* $\mathbb{1}^\lambda$ *outputs a prime* p*, the description of a cyclic group* $\mathbb{G}$ *of order* p *and two elements* $g, h \in \mathbb{G} \setminus \{1\}$*. For any function* $S(\lambda)$*, we say that the* $S(\lambda)$*-uniform decisional Diffie-Hellman assumption holds for* DDHGen *if no uniform adversary running in* $\mathsf{poly}\big(\lambda, S(\lambda)\big)$ *time can distinguish between the following distributions*

$$\Big\{g^r, h^r \Big| r \xleftarrow{\$} [p]\Big\} \qquad \Big\{g^r, h^s \Big| r, s \xleftarrow{\$} [p]\Big\}$$

Definition 41 (Post-Quantum Uniform LWE). *Let* m, n, q *be functions in the security parameter. Let* χ *be a distribution over* $\mathbb{Z}_q^m$ *that is efficiently samplable using a uniform algorithm and let* $B(\lambda)$ *be a bound on* $\|\chi(\mathbb{1}^\lambda)\|$*. Let* LWEGen *be a uniform, polynomial time, deterministic algorithm that on input the security parameter* $\mathbb{1}^\lambda$ *outputs a rank-n matrix* $A \in \mathbb{Z}_q^{m \times n}$*. For any function* $S(\lambda)$*, we say that the post-quantum* $S(\lambda)$*-uniform LWE assumption holds for* LWEGen *if no quantum, uniform adversary running in* $\mathsf{poly}\big(\lambda, S(\lambda)\big)$ *time can distinguish between the following distributions*

$$\Big\{A^\intercal \cdot s + x \Big| s \xleftarrow{\$} \mathbb{Z}_q^n, x \xleftarrow{\$} \chi\Big\} \qquad \Big\{v \Big| v \xleftarrow{\$} \mathbb{Z}_q^m\Big\}$$

We say that LWEGen *is trapdoored if, for every* $\lambda \in \mathbb{N}$*, there exists* $u \in \mathbb{Z}_q^m$ *such that* $u_0 = 1$*,* $\|u\| \cdot B(\lambda) < \lfloor q/4 \rfloor$ *and* $A \cdot u = 0$ *where* $A = \mathsf{LWEGen}(\mathbb{1}^\lambda)$*.*

Defining challengeless one-way functions. Our first simulation-sound construction relies on a new notion called *challengeless one-way function.* The concept is a natural adaptation of one-way function to the uniform setting. The main difference is that security is not ensured by randomness, but leveraging the computational limits of uniform adversaries. Informally, a challengeless one-way function is a uniform deterministic algorithm COWF that, on input any value u, it either accepts it or rejects it. We require that, for every uniform, computationally bounded adversary, it is hard to find an accepting input. A natural example of challengeless one-way function is the one that checks if a given pair (x_0, x_1) consists of a collision for a keyless hash function.

Definition 42 (Challengeless one-way function). *Let* $p(\lambda)$ *be a polynomial function. A challengeless one-way function with input size* $p(\lambda)$ *is a uniform deterministic polynomial-time algorithm* COWF *that takes as input the security parameter* $\mathbb{1}^\lambda$ *and a value* $u \in \{0,1\}^{p(\lambda)}$*. We require the following properties:*

1. *For every* $\lambda \in \mathbb{N}$*, there exists at least one element* u_λ *such that* $\mathsf{COWF}(\mathbb{1}^\lambda, u_\lambda) = 1$.
2. *For every uniform PPT algorithm* $\mathcal{A}$

$$\Pr\Big[\mathsf{COWF}(\mathbb{1}^\lambda, u) = 1 \Big| u \xleftarrow{\$} \mathcal{A}(\mathbb{1}^\lambda)\Big] = \mathsf{negl}(\lambda).$$

If there exists a unique u_λ such that $\mathsf{COWF}(1\!\!1^\lambda, u_\lambda) = 1$ *for every* $\lambda \in \mathbb{N}$*, we say that the challengeless one-way function is injective.*

Let $S(\lambda)$ *be a function of the security parameter. We say that* COWF *is* $S(\lambda)$*-secure if, for every uniform algorithm* $\mathcal{A}$ *running in* $\mathsf{poly}\big(\lambda, S(\lambda)\big)$ *time,*

$$\Pr\Big[\mathsf{COWF}(1\!\!1^\lambda, u) = 1 \Big| u \xleftarrow{\$} \mathcal{A}(1\!\!1^\lambda, 1\!\!1^{S(\lambda)})\Big] = \mathsf{negl}(\lambda).$$

We say that COWF *is post-quantumly* $S(\lambda)$*-secure if, for every uniform quantum algorithm* $\mathcal{Q}$ *running in* $\mathsf{poly}\big(\lambda, S(\lambda)\big)$ *time,*

$$\Pr\Big[\mathsf{COWF}(1\!\!1^\lambda, u) = 1 \Big| u \xleftarrow{\$} \mathcal{Q}(1\!\!1^\lambda, 1\!\!1^{S(\lambda)})\Big] = \mathsf{negl}(\lambda).$$

We observe that uniform DDH and uniform LWE immediately lead to challengeless one-way functions that are subexponentially secure. In the case of uniform DDH, the construction is post-quantumly broken and injective. In the case of uniform LWE instead, we also obtain security against uniform quantum adversaries. These properties will be fundamental in our first simulation-sound U-NIZK.

Theorem 20. *Let* $S(\lambda)$ *be a function of the security parameter. If* $S(\lambda)$*-uniform DDH holds for* DDHGen*, then the algorithm that, on input* $u \in \mathbb{Z}_p$*, checks if* $h = g^u$ *for* $(p, \mathbb{G}, g, h) \leftarrow \mathsf{DDHGen}(1\!\!1^\lambda)$ *is an* $S(\lambda)$*-secure injective challengeless OWF.*

Theorem 21. *Let* $S(\lambda)$ *be a function of the security parameter. If the post-quantum* $S(\lambda)$*-uniform LWE assumption holds for* LWEGen *and the latter is trapdoored, then the algorithm that, on input* $u \in \mathbb{Z}_q^m$*, checks if* $A \cdot u = 0$*,* $u \neq 0$ *and* $\|u\| \cdot B(\lambda) < \lfloor q/4 \rfloor$ *for* $A \leftarrow \mathsf{LWEGen}(1\!\!1^\lambda)$ *is a post-quantumly* $S(\lambda)$*-secure challengeless OWF.*

Independently hard challengeless one-way functions. Challengeless one-way functions can be tricky objects: leaking values accepted by one such construction can, in principle, compromise the security of other challengeless one-way functions[16]. Luckily, assuming the subexponential hardness of uniform-DDH, the schemes we introduced above do not suffer from this issue. Specifically, we can parametrise the security of the constructions so that the uniform-DDH construction is $S(\lambda)$-secure while it is possible to brute-force the uniform-LWE construction in time $S(\lambda)$. This implies that the first challengeless one-way function is secure even if we leak values accepted by the second one. The opposite holds too due to the post-quantum security of uniform LWE: a quantum adversary can easily find values accepted by the first construction while having no chance against the second one. If a pair of challengeless one-way functions behaves in this way, we say that they are *independently hard.*

[16] Consider, e.g., the construction based on uniform-DDH and a similar one which accepts u if and only if $g^u = h^2$.

Definition 43 (Independently hard challengeless one-way functions). *Let* COWF_0 *and* COWF_1 *be challengeless one-way functions. We say that* COWF_0 *and* COWF_1 *are independently hard if, for every* $\lambda \in \mathbb{N}$*, there exist values* $u_{\lambda,0}, u_{\lambda,1}$ *such that, for every* $b \in \{0,1\}$*,*

- $\mathsf{COWF}_b(\mathbb{1}^\lambda, u_{\lambda,b}) = 1$.
- *For every uniform PPT adversary* $\mathcal{A}$*,*

$$\Pr\Big[\mathsf{COWF}_b(\mathbb{1}^\lambda, u) = 1 \Big| u \xleftarrow{\$} \mathcal{A}(\mathbb{1}^\lambda, u_{\lambda,1-b})\Big] = \mathsf{negl}(\lambda).$$

Let $S(\lambda)$ *be a function of the security parameter. We say that* COWF_0 *and* COWF_1 *are* $S(\lambda)$*-independently hard if the above property holds even against adversaries* $\mathcal{A}$ *running in* $\mathsf{poly}\big(\lambda, S(\lambda)\big)$ *time.*

We will use independently hard challengeless-one-way functions to build simulation-sound U-NIZKs that do not need CRSs. As we explained above, we can obtain what we need from uniform-DDH and uniform-LWE.

Theorem 22. *Let* $S(\lambda) = 2^{\lambda^{O(1)}}$ *be a function of the security parameter. Assuming the subexponential security of uniform-DDH and the subexponential, post-quantum security of uniform-LWE, there exists a pair of* $S(\lambda)$*-independently hard challengeless one-way functions.*

The above result has clearly the disadvantage that the resulting pair of challengeless one-way functions is not secure against quantum adversaries, however, there may be other solutions that do not suffer from this issue.

Challengeless labelled one-way functions. The second simulation-sound U-NIZK that we present relies on a stronger assumption, namely the existence of *challengeless labelled one-way functions.* These correspond to particular challengeless one-way functions that, in addition to the usual input, receive also a label id. We also require the existence of a trapdoor u that, for any label id, allows efficiently determining a value u^{id} such that $(u^{\mathsf{id}}, \mathsf{id})$ is an accepting pair. Security requires that no uniform, computationally bounded adversary can find an accepting pair $(\widehat{u}, \widehat{\mathsf{id}})$ even if it has access to an oracle that, on input any labelled id, returns an accepting pair $(u^{\mathsf{id}}, \mathsf{id})$.

Definition 44 (Challengeless labelled one-way function). *An challengeless labelled one-way function is a pair of uniform PPT algorithms* $(\mathsf{CLOWF}, \mathsf{Derive})$ *with the following properties:*

1. CLOWF *is deterministic.*
2. *For every* $\lambda \in \mathbb{N}$*, there exits a value* u_λ *such that* $\mathsf{CLOWF}(\mathbb{1}^\lambda, \mathsf{Derive}(\mathbb{1}^\lambda, u_\lambda, \mathsf{id}), \mathsf{id}) = 1$ *with probability* 1 *for every identity* id.
3. *For every uniform PPT algorithm* $\mathcal{A}$*,*

$$\Pr\Big[\mathsf{id} \notin Q, \mathsf{CLOWF}(\mathbb{1}^\lambda, u^{\mathsf{id}}, \mathsf{id}) = 1 \Big| (\mathsf{id}, u^{\mathsf{id}}) \xleftarrow{\$} \mathcal{A}^{\mathsf{Derive}(\mathbb{1}^\lambda, u_\lambda, \cdot)}(\mathbb{1}^\lambda)\Big] = \mathsf{negl}(\lambda),$$

where Q *denotes the set of identities queried by* $\mathcal{A}$ *to* $\mathsf{Derive}(\mathbb{1}^\lambda, u_\lambda, \cdot)$.

Let $B(\lambda)$ be a function of the security parameter. We say that the challengeless labelled one-way function is $B(\lambda)$-bounded if, for every $\lambda \in \mathbb{N}$ and identity id*,*

$$\left|\left\{v \middle| \mathsf{CLOWF}(\mathbb{1}^\lambda, v, \mathsf{id}) = 1\right\}\right| \leq B(\lambda).$$

We say that the construction is injective if it is 1-bounded.

For any function $S(\lambda)$, we say that the construction is $S(\lambda)$-secure if, for every uniform algorithm $\mathcal{A}$ running in $\mathsf{poly}\big(\lambda, S(\lambda)\big)$ *time,*

$$\Pr\left[\mathsf{id} \notin Q, \mathsf{CLOWF}(\mathbb{1}^\lambda, u^{\mathsf{id}}, \mathsf{id}) = 1 \middle| (\mathsf{id}, u^{\mathsf{id}}) \xleftarrow{\$} \mathcal{A}^{\mathsf{Derive}(\mathbb{1}^\lambda, u_\lambda, \cdot)}(\mathbb{1}^\lambda, \mathbb{1}^{S(\lambda)})\right] = \mathsf{negl}(\lambda).$$

Building challengeless labelled one-way functions is clearly harder than building simple challengeless one-way functions. In this paper, we present only heuristic constructions. We take inspiration from the identity-based encryption literature [CHK03,BB04,Wat05,Gen06]. We notice indeed that the secret-keys of IBE schemes have the kind of structure we are looking for: given a decryption key for any identity we can efficiently validate it. Moreover, even given access to a key generation oracle, it is hard to compute decryption keys for non-queried identities. Finally, there exists a trapdoor that allows us to efficiently retrieve the decryption key for any identity. If the master public key mpk of the IBE scheme is statistically close to uniform, we can therefore hope that a SHA hash function allows us to deterministically generate mpk without disclosing the secret counterpart.

Another candidate construction, inspired by [Gen06], is to deterministically generate the parameters of a pairing-based bilinear map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ using a SHA hash function along with values g, g_1 and h in $\mathbb{G}$. In this way, we can hope that the values u and v such that $g_1 = g^u$ and $h = g^v$ remain hidden to uniform adversaries. The trapdoor of the labelled one-way function will be u. For each identity id, the only accepted value is $u^{\mathsf{id}} := \big(h \cdot g^{-f(\mathsf{id})}\big)^{1/(u-\mathsf{id})}$ where f denotes another SHA hash function. We can easily verify u^{id} by checking if $e(g_1/g^{\mathsf{id}}, u^{\mathsf{id}}) = e(h \cdot g^{-f(\mathsf{id})}, g)$. Observe that if the SHA hash functions were modelled as random oracles, an attack to the labelled one-way function would lead to an attack to the IBE scheme of [Gen06]. A nice property of this candidate construction is its injectivity and the fact that is broken against quantum adversaries. We will rely on these characteristics in Section 10.

A final option is to use signature schemes in which the verification key is statistically close to uniform. Using a SHA hash function we deterministically generate a verification key vk. We can hope that the private counterpart, corresponding to the trapdoor of the labelled one-way function, is hard to compute for every uniform adversary. The value u^{id} associated to the identity id will consist of a signature on id under vk. Clearly, u^{id} can be easily verified. Moreover, thanks to the security of the signature, we can hope that revealing u^{id} does not help in finding u^{id^*} for any $\mathsf{id}^* \neq \mathsf{id}$. Observe that if we consider a deterministic signature scheme, this candidate labelled one-way function becomes injective. Furthermore, if we use DLOG-based constructions such as EdDSA or ECDSA, the labelled one-way function is broken against quantum adversaries.

The first simulation-sound U-NIZK. We now present our first simulation-sound U-NIZK. We make use of the same primitives as in the restricted simulation-sound construction in Section 9.1 together with the subexponential hardness of uniform DDH and post-quantum uniform LWE.

The reason why the construction in Section 9.1 failed to achieve full simulation-soundness is the fact that witness indistinguishability only guarantees that the proof does not leak any information allowing distinguishing which witness was used. Specifically, if the statement belongs to the language, we are sure that a simulated proof does not leak the trapdoor. However, if the statement is not in the language, all the witnesses are based on the same trapdoor (the value hidden in the commitment), so leaking it would not compromise witness indistinguishability. In order to achieve full-simulation soundness, we will therefore use two trapdoors: a pair of accepting values for two independently hard challengeless one-way functions COWF_0 and COWF_1. In other words, each of them will retain their security properties even if we leak values accepted by the other one.

The idea is to include two CCA commitments c_0 and c_1 of 0 in the proof instead of just one. The NIWI will prove that either our statement lies in the language or there exists $b \in \{0,1\}$ such that c_b is a commitment to a value accepted by COWF_b. As for the construction in Section 9.1, we will sign the NIWI and the commitments using a strong one-time signature. The tag used in the CCA commitments will be the verification key for such signature.

In order to simulate a proof, we will always commit to a value u_0 accepted by COWF_0. The proof of zero-knowledge will proceed similarly to the restricted simulation-sound construction. As for simulation-soundness, the idea is the following. Thanks to independent hardness, if the oracle uses a value accepted by COWF_0 as trapdoor for the simulated proofs, the adversary cannot generate valid proofs for false statements using values accepted by COWF_1. For the same reason, but in a symmetric way, if the oracle uses a value accepted by COWF_1 as trapdoor for the simulated proofs, the adversary cannot generate valid proofs for false statements using values accepted by COWF_0. We conclude by relying on subexponential witness-indistinguishability: we show that it is impossible to tell if the simulated proofs used COWF_0 or COWF_1 for the trapdoor even if we have enough computational power to break the hiding properties of the commitments. That guarantees that the adversary cannot use either of the challengeless one-way functions to forge proofs.

Formalising the construction. Let $T(\lambda)$ be a function of the security parameter and let $e \in \mathbb{N}$ be such that $T(\lambda) < 2^{\lambda^e}$. We rely on a e-computation enabled non-interactive CCA commitment [GKLW21]. We require that the construction satisfies perfect correctness. Let $S(\lambda)$ denote the running time of $\mathsf{CCACom.Val}$.

Let COWF_0 and COWF_1 be $\max\{T(\lambda), S(\lambda)\}$-independently hard, challengeless one-way functions with input size $p(\lambda)$. For every $b \in \{0,1\}$ and $\lambda \in \mathbb{N}$, let $u_{\lambda,b}$ be the value accepted by COWF_b that could be leaked without compromising the security of COWF_{1-b}. We require that the sequence $(u_{\lambda,0}, u_{\lambda,1})_{\lambda\in\mathbb{N}}$ is 2^{λ^e}-computable.

Let NIWI be a perfectly sound, witness indistinguishable proof system for the relation $\mathcal{R}_{\mathsf{NIWI}}$ defined below

$$\left\{ \begin{matrix} (\mathsf{vk}, c_0, c_1, x), \\ w \end{matrix} \middle| (x, w) \in \mathcal{R} \quad \text{OR} \quad \exists b \in \{0,1\} \text{ s.t. } \begin{pmatrix} w = (u_b, r) \\ \mathsf{COWF}_b(\mathbb{1}^\lambda, u_b) = 1 \\ c_b = \mathsf{CCACom}\big(\mathbb{1}^\lambda, \mathsf{vk}, u_b;\ r\big) \end{pmatrix} \right\}$$

We require that NIWI is witness indistinguishable even against adversaries running in $\mathsf{poly}\big(\lambda, S(\lambda)\big)$ time. Finally, let $\mathsf{SOTS} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ be a strong one-time signature. To summarise, we have $T(\lambda), S(\lambda) \ll 2^{\lambda^e}$.

```
A simulation-sound NIZK without CRS
$\mathsf{Prove}(\mathbb{1}^\lambda, x, w)$

  1. $(\mathsf{vk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{SOTS.Gen}(\mathbb{1}^\lambda)$
  2. $c_0 \xleftarrow{\$} \mathsf{CCACom}(\mathbb{1}^\lambda, \mathsf{vk}, 0^{p(\lambda)})$
  3. $c_1 \xleftarrow{\$} \mathsf{CCACom}(\mathbb{1}^\lambda, \mathsf{vk}, 0^{p(\lambda)})$
  4. $\pi' \xleftarrow{\$} \mathsf{NIWI.Prove}\big(\mathbb{1}^\lambda, (\mathsf{vk}, c_0, c_1, x), w\big)$
  5. $s \leftarrow \mathsf{SOTS.Sign}\big(\mathsf{sk}, (\mathsf{vk}, c_0, c_1, \pi')\big)$
  6. Output $\pi := (\mathsf{vk}, c_0, c_1, \pi', s)$

$\mathsf{Verify}\big(\pi = (\mathsf{vk}, c_0, c_1, \pi', s), x\big)$

  1. $b_0 \leftarrow \mathsf{SOTS.Verify}\big(\mathsf{vk}, (\mathsf{vk}, c_0, c_1, \pi'), s\big)$
  2. $b_1 \leftarrow \mathsf{NIWI.Verify}\big(\pi', (\mathsf{vk}, c_0, c_1, x)\big)$
  3. Output $b_0 \wedge b_1$.

$\mathsf{SimSetup}(\mathbb{1}^\lambda)$

  1. Get $u_{\lambda,0}$ where $\mathsf{COWF}(\mathbb{1}^\lambda, u_{\lambda,0}) = 1$ as auxiliary input.
  2. Output the empty string along with $\tau = u_{\lambda,0}$.

$\mathsf{SimProve}(\tau, x)$

  1. $(\mathsf{vk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{SOTS.Gen}(\mathbb{1}^\lambda)$
  2. $r \xleftarrow{\$} \{0,1\}^{q(\lambda)}$
  3. $c_0 \leftarrow \mathsf{CCACom}(\mathbb{1}^\lambda, \mathsf{vk}, \tau;\ r)$
  4. $c_1 \xleftarrow{\$} \mathsf{CCACom}(\mathbb{1}^\lambda, \mathsf{vk}, 0^{p(\lambda)})$
  5. $\pi' \xleftarrow{\$} \mathsf{NIWI.Prove}\big(\mathbb{1}^\lambda, (\mathsf{vk}, c_0, c_1, x), (\tau, r)\big)$
  6. $s \leftarrow \mathsf{SOTS.Sign}\big(\mathsf{sk}, (\mathsf{vk}, c_0, c_1, \pi')\big)$
  7. Output $\pi := (\mathsf{vk}, c_0, c_1, \pi', s)$
```

Fig. 46. A simulation-sound NIZK without CRS

Theorem 23. *Let $a := (a_\lambda)_{\lambda \in \mathbb{N}}$ be a T-computable sequence. Assume the existence of computation-enabled CCA commitments with* $\mathsf{poly}(\lambda)$ *tag size. Fur-*

thermore, assume the existence of subexponentially, independently hard, challengeless one-way functions. Finally, assume the existence of strong one-time signatures and subexponentially secure non-interactive witness-indistinguishable proofs without CRS. Then, the construction in Fig. 46 is an a-disclosed simulation-sound NIZK without CRS for $\mathcal{R}$ with security against uniform PPT adversaries.

Proof. Completeness follows immediately from the completeness of NIWI.

Claim. The construction in Fig. 46 is an a-disclosed multi-theorem zero-knowledge U-NIZK for $\mathcal{R}$.

Proof of the claim. Let $M(\lambda)$ be a polynomial upper-bounding the number of tuples (x_i, w_i) queried by $\mathcal{A}$. Since $\mathcal{A}$ is PPT, we know that M exists. Let π_i denote the answer to the i-th query.

For every $i \in [M] \cup \{0\}$, we define Hybrid i as the hybrid in which, for every $j \leq i$, we generate π_j using $\mathsf{SimProve}(\tau, x_j)$. For very $j > i$ instead, we generate π_j using $\mathsf{Prove}(1\!\!1^\lambda, x_j, w_j)$.

Notice that Hybrid 0 is identical to the zero-knowledge game when $b = 0$. Hybrid M is instead identical to the zero-knowledge game when $b = 1$. In order to prove our claim, it is sufficient to prove that no uniform PPT adversary can distinguish between Hybrid $i - 1$ and Hybrid i for a randomly sampled $i \overset{\$}{\leftarrow} [M]$.

We show this by relying on a sequence of indistinguishable subhybrids.

Hybrid' 0. This hybrid coincides with Hybrid $i - 1$ for $i \overset{\$}{\leftarrow} [M]$.

Hybrid' 1. In this hybrid, we change the proof π_i. In particular, instead of committing to $0^{p(\lambda)}$, we commit to the value $\tau = u_{\lambda,0}$ output by $\mathsf{SimSetup}$. All the rest remains as in Hybrid' 0.

This hybrid is indistinguishable from the previous one by the e-computation enabled CCA hiding property of CCACom. In the reduction, the adversary $\mathcal{B}$ starts its execution by querying the Turing machine P that computes $(u_{\lambda,0}, u_{\lambda,1}, a_\lambda)$. Then, it samples $i \overset{\$}{\leftarrow} [M]$ and runs $\mathcal{A}$. It replies to the first $i - 1$ queries using $\mathsf{SimProve}$. Starting from the $(i + 1)$-th query, it instead uses Prove.

The adversary $\mathcal{B}$ deviates from the game in the i-th Prove query. It begins by generating the signature key pair $(\mathsf{vk}, \mathsf{sk})$ and queries $(\mathsf{tag} = \mathsf{vk}, 0^{p(\lambda)}, u_{\lambda,0})$ to its challenger. It then uses the answer c_0 and $(\mathsf{vk}, \mathsf{sk})$ to generate π_i. The adversary $\mathcal{B}$ terminates the execution outputting the same value that $\mathcal{A}$ generates. Notice that if $\mathcal{A}$ distinguishes between Hybrid' 0 and Hybrid' 1, then $\mathcal{B}$ is a e-computation enabled adversary breaking the hiding properties of CCACom.

Hybrid' 2. In this hybrid, we change again the proof π_i. In particular, instead of using w_i as witness for NIWI, we use τ and the randomness used to commit to it. This hybrid is indistinguishable from the previous one by the S-witness indistinguishability of NIWI.

Notice that NIWI is secure against non-uniform adversaries so, in the reduction, we can assume that the adversary is given $\tau = u_{\lambda,0}$ and a_λ as part of its auxiliary input. We build a PPT adversary that breaks the witness indistinguishability of NIWI. It starts its execution sampling a random $i \overset{\$}{\leftarrow} [M]$ and it simulates the game as in Hybrid' 1 to an internal copy of $\mathcal{A}$. It deviates from the game at the i-th Prove query. In particular, after generating vk, c_0 and c_1 as usual, it queries $(\mathsf{vk}, c_0, c_1, x_i)$ along with the witnesses w_i and (τ, r) to the NIWI challenger. Here, r denotes the randomness used for the generation of c_0. The answer π' is included in the proof returned to $\mathcal{A}$. At the end of its execution, $\mathcal{B}$ outputs the same bit as $\mathcal{A}$. Notice that if $\mathcal{A}$ succeeds in distinguishing, $\mathcal{B}$ succeeds too.

Observe that Hybrid' 2 is identical to Hybrid i for a random $i \xleftarrow{\$} [M]$. This ends the proof of the claim. ■

Claim. The construction in Fig. 46 is an a-disclosed simulation-sound U-NIZK for $\mathcal{R}$.

Proof of the claim. We use a hybrid argument.

Hybrid 0. In this hybrid, we provide the adversary with oracle access to $\mathsf{SimProve}$. The challenger outputs 1 when the adversary terminates its execution outputting an accepting proof. Let Q denote the set of the oracle responses.

Hybrid 1. In this hybrid, when the adversary generates an accepting proof $\pi = (\mathsf{vk}, c_0, c_1, \pi', s)$ for a false statement x, the challenger checks whether vk coincides with the verification key in one of the responses of the simulation oracle. In that case, the challenger outputs 0, in the other cases it behaves as before.

Observe that Hybrid 0 and Hybrid 1 are indistinguishable due to the security of the strong one-time signature. Indeed, the adversary can distinguish only if it generates an accepting proof $\pi = (\mathsf{vk}, c_0, c_1, \pi', s)$ for a false statement x such that $(\pi, x) \notin Q$ but vk was previously generated by $\mathsf{SimProve}$. In other words, $\mathcal{A}$ would produce a forgery with non-negligible probability. Notice that the one-time signature is secure against non-uniform adversaries, so in the reduction, we can assume that the adversary receives a_λ as part of its advice string.

Hybrid 2. In this hybrid, when the adversary generates an accepting proof $\pi = (\mathsf{vk}, c_0, c_1, \pi', s)$ for a false statement x, the challenger checks whether $\mathsf{COWF}_1(\mathbb{1}^\lambda, u_1) = 1$ where $u_1 = \mathsf{CCACom.Val}(\mathsf{vk}, c_1)$. In that case, the challenger outputs 0, in the other cases it behaves as before.

Hybrid 2 and Hybrid 1 are indistinguishable under the $\max\{T(\lambda), S(\lambda)\}$-independent hardness of COWF_0 and COWF_1. Indeed, an adversary can distinguish only if it generates a proof where

$$\mathsf{COWF}_1\big(\mathbb{1}^\lambda, \mathsf{CCACom.Val}(\mathsf{vk}, c_1)\big) = 1.$$

In the reduction, we build a uniform PPT adversary running in $\mathsf{poly}\big(\lambda, T(\lambda), S(\lambda)\big)$ time that retrieves a value u_1 such that $\mathsf{COWF}_1(\mathbb{1}^\lambda, u_1) = 1$ with non-negligible probability. The adversary $\mathcal{B}$ starts its execution by retrieving $\tau = u_{\lambda,0}$, which is given by the challenger, and a_λ. This operation requires $\mathsf{poly}\big(\lambda, T(\lambda)\big)$ time. Then, it proceeds by running an internal copy of $\mathcal{A}$ simulating $\mathsf{SimProve}$ using τ. When $\mathcal{A}$ outputs a proof $\pi = (\mathsf{vk}, c, \pi', s)$, $\mathcal{B}$ outputs $\mathsf{CCACom.Val}(\mathsf{vk}, c_1)$. With non-negligible probability, the latter coincides with a value u_1 such that $\mathsf{COWF}_1(\mathbb{1}^\lambda, u_1) = 1$. Observe that $\mathcal{B}$ can run $\mathsf{CCACom.Val}(\mathsf{vk}, c)$ in $\mathsf{poly}\big(\lambda, S(\lambda)\big)$.

Now, let $M(\lambda)$ be a polynomial upper-bounding the number of queries to $\mathsf{SimProve}$ issued by the adversary. For every $i \in [M] \cup \{0\}$, we define the following hybrids.

Hybrid 3.*i*. In this hybrid, in the first i simulated proofs $\pi = (\mathsf{vk}, c_0, c_1, \pi', s)$, c_1 will be a commitment to $u_{\lambda,1}$. In the remaining simulated proofs, c_1 will be a commitment to $0^{p(\lambda)}$. The rest remains as in Hybrid 2.

Observe that Hybrid 3.0 is identical to Hybrid 2. We now show that no uniform PPT adversary can distinguish between Hybrid 3.i and Hybrid 3.$(i-1)$ for a random $i \xleftarrow{\$} [M]$ due to the e-computation enable CCA hiding property of CCACom. In the reduction, we build a uniform PPT adversary $\mathcal{B}$ that samples $i \xleftarrow{\$} [M]$ and queries the Turing machine P that generates the triple $(u_{\lambda,0}, u_{\lambda,1}, a_\lambda)$. The adversary $\mathcal{B}$ proceeds by simulating the game as in Hybrid 3.$(i-1)$ to an internal copy of $\mathcal{A}$ using $(u_{\lambda,0}, u_{\lambda,1})$. It deviates from the game at the i-th simulation query. Specifically, after generating the signature key pair $(\widehat{\mathsf{vk}}, \widehat{\mathsf{sk}})$, it queries $(\mathsf{tag} = \widehat{\mathsf{vk}}, 0^{p(\lambda)}, u_{\lambda,1})$ to its challenger. It uses the answer $\widehat{c_1}$ to generate the rest of the simulated proof. When $\mathcal{A}$ outputs a proof $\pi = (\mathsf{vk}, c_0, c_1, \pi', s)$ and a statement x, $\mathcal{B}$ can check the value hidden in c_1 by querying it to $\mathsf{CCACom.Val}$. Observe that if $\mathsf{vk} = \widehat{\mathsf{vk}}$, $\mathcal{B}$ can simply provide $\mathcal{A}$ with 0. At the end, $\mathcal{B}$ outputs the same bit as $\mathcal{A}$. So, if $\mathcal{A}$ succeeds in distinguishing, $\mathcal{B}$ succeeds too.

Hybrid 4.*i*. In this hybrid, in the first i simulated proofs $\pi = (\mathsf{vk}, c_0, c_1, \pi', s)$, π' uses $u_{\lambda,1}$ and the randomness used in c_1 as witness. In the remaining simulated proofs, π' uses $u_{\lambda,0}$ and the randomness used in c_0. The rest remains as in Hybrid 3.M.

Observe that Hybrid 4.0 is identical to Hybrid 3.M. We now show that no uniform PPT adversary can distinguish between Hybrid 4.i and Hybrid 4.$(i-1)$ for a random $i \xleftarrow{\$} [M]$ due to the non-uniform witness-indistinguishability of NIWI. In the reduction, we build a non-uniform adversary $\mathcal{B}$ running in $\mathsf{poly}\big(\lambda, S(\lambda)\big)$ time. It starts its execution by receiving the triple $(u_{\lambda,0}, u_{\lambda,1}, a_\lambda)$ as advice string. Then, it samples $i \xleftarrow{\$} [M]$ and simulates the game as in Hybrid 4.$(i-1)$ to an internal copy of $\mathcal{A}$ using $(u_{\lambda,0}, u_{\lambda,1})$. It deviates from the game at the i-th simulation query. Specifically, after generating $\widehat{c_0}$ and $\widehat{c_1}$ with randomness r_0 and r_1 respectively, it queries the statement $(\widehat{\mathsf{vk}}, \widehat{c_0}, \widehat{c_1}, \widehat{x})$ and the witnesses $(u_{\lambda,0}, r_0)$ and $(u_{\lambda,1}, r_1)$ to its challenger. It uses the answer $\widehat{\pi}'$ to generate the rest of the simulated proof. When $\mathcal{A}$ outputs a proof $\pi = (\mathsf{vk}, c_0, c_1, \pi', s)$ and a statement x, $\mathcal{B}$ runs $\mathsf{CCACom.Val}(\mathsf{vk}, c_1)$. The operation requires $\mathsf{poly}\big(\lambda, S(\lambda)\big)$ time. At the end, $\mathcal{B}$ outputs the same bit as $\mathcal{A}$. So, if $\mathcal{A}$ succeeds in distinguishing, $\mathcal{B}$ succeeds too.

Hybrid 5.*i*. In this hybrid, in the first i simulated proofs $\pi = (\mathsf{vk}, c_0, c_1, \pi', s)$, c_0 will be a commitment to $0^{p(\lambda)}$. In the remaining simulated proofs, c_0 will be a commitment to $u_{\lambda,0}$. The rest remains as in Hybrid 4.M.

Observe that Hybrid 5.0 is identical to Hybrid 4.M. We now show that no uniform PPT adversary can distinguish between Hybrid 5.i and Hybrid 5.$(i-1)$ for a random $i \xleftarrow{\$} [M]$ due to the e-computation enable CCA hiding property of CCACom. In the reduction, we build a uniform PPT adversary $\mathcal{B}$ that samples $i \xleftarrow{\$} [M]$ and queries the Turing machine P that generates the triple $(u_{\lambda,0}, u_{\lambda,1}, a_\lambda)$. The adversary $\mathcal{B}$ proceeds by simulating the game as in Hybrid 5.$(i-1)$ to an internal copy of $\mathcal{A}$ using $(u_{\lambda,0}, u_{\lambda,1})$. It deviates from the game at the i-th simulation query. Specifically, after generating the signature key pair $(\widehat{\mathsf{vk}}, \widehat{\mathsf{sk}})$, it queries $(\mathsf{tag} = \widehat{\mathsf{vk}}, u_{\lambda,0}, 0^{p(\lambda)})$ to its challenger. It uses the answer $\widehat{c_0}$ to generate the rest of the simulated proof. When $\mathcal{A}$ outputs a proof $\pi = (\mathsf{vk}, c_0, c_1, \pi', s)$ and a statement x, $\mathcal{B}$ can check the value hidden in c_1 by querying it to $\mathsf{CCACom.Val}$. Observe that if $\mathsf{vk} = \widehat{\mathsf{vk}}$, $\mathcal{B}$ can simply provide $\mathcal{A}$ with 0. At the end, $\mathcal{B}$ outputs the same bit as $\mathcal{A}$. So, if $\mathcal{A}$ succeeds in distinguishing, $\mathcal{B}$ succeeds too.

Hybrid 6. In this hybrid, when the adversary generates an accepting proof $\pi = (\mathsf{vk}, c_0, c_1, \pi', s)$ for a false statement x, the challenger checks whether $\mathsf{COWF}_0(\mathbb{1}^\lambda, u_0) = 1$ or $\mathsf{COWF}_1(\mathbb{1}^\lambda, u_1) = 1$ where $u_0 = \mathsf{CCACom.Val}(\mathsf{vk}, c_0)$ and $u_1 = \mathsf{CCACom.Val}(\mathsf{vk}, c_1)$. In that case, the challenger outputs 0, in the other cases it behaves as before.

Hybrid 6 is indistinguishable from Hybrid 5.M under the $\max\{T(\lambda), S(\lambda)\}$-independent hardness of COWF_0 and COWF_1. Indeed, if $\mathcal{A}$ distinguishes, then it must be able to generate a proof such that

$$\mathsf{COWF}_0\big(\mathbb{1}^\lambda, \mathsf{CCACom.Val}(\mathsf{vk}, c_0)\big) = 1.$$

In the reduction, we build a uniform adversary $\mathcal{B}$ running in $\mathsf{poly}\big(\lambda, T(\lambda), S(\lambda)\big)$ time. The adversary $\mathcal{B}$ starts its execution by recovering $u_{\lambda,1}$, which is given by its challenger, and a_λ. Then, it simulates the game as in Hybrid 5.M to an internal copy of $\mathcal{A}$ using $u_{\lambda,1}$. When the adversary outputs a proof $\pi = (\mathsf{vk}, c_0, c_1, \pi', s)$, $\mathcal{B}$ outputs $u_0 = \mathsf{CCACom.Val}(\mathsf{vk}, c_0)$. With non-negligible probability $\mathsf{COWF}_0(\mathbb{1}^\lambda, u_0) = 1$. Notice that the last operation requires $\mathsf{poly}\big(\lambda, S(\lambda), T(\lambda)\big)$ time. We reached a contradiction.

Observe that in Hybrid 6, by the perfect soundness of NIWI and the perfect correctness of CCACom, the challenger always outputs 0 when it is provided with a proof for a false statement. That means that, in Hybrid 0, the adversary could not generate valid proofs for false statements except with negligible probability. This ends the proof. ■

□

The second simulation-sound U-NIZK. We now present our second simulation-sound U-NIZK. The construction is simpler but relies on challengeless labelled one-way functions, a primitive for which, currently, we have only heuristic instantiations.

Once again, the scheme follows the blueprint of the construction in Section 9.1. The main difference is that the simulation trapdoor will coincide with the trapdoor of the challengeless labelled one-way function. As before, the proof will consist of a commitment c to 0 along with a NIWI proof. The latter will guarantee that either our statement x lies in the language or c is a commitment to a value u^{id} that is accepted by the challengeless labelled OWF. The label id

is a strong one-time signature verification key vk. As in Section 9.1, the NIWI proof and c are signed using vk.

To simulate a proof, we just commit to u^{id} and we use the latter as witness for the NIWI. Zero-knowledge is guaranteed by the hiding properties of the commitment and witness indistinguishability. As for simulation-soundness, the adversary cannot reuse the same verification keys as the simulation oracle; it is forced to craft its own proof using a fresh key pair. Hence, even if the NIWI leaks any trapdoor u^{id} used in the simulated proofs, by the security of the labelled one-way function, the adversary will not be able to derive the trapdoor corresponding to its verification key. With the current approach, we will be able to achieve security even if the commitment is malleable. We will therefore obtain a U-NIZK without CRS that achieves multi-theorem zero-knowledge even against non-uniform adversaries. Soundness will clearly be restricted to the uniform setting.

Formalising the construction. Let $T(\lambda)$ be a function of the security parameter. We rely on a perfectly binding, computationally hiding non-interactive commitment scheme Com. We also require that the value hidden in the commitments can be retrieved with probability 1 in uniform $\mathsf{poly}\big(\lambda, S(\lambda)\big)$ time.

We make use of a challengeless labelled one-way CLOWF functions that is $(T+S)$-secure.

Let NIWI be a perfectly sound, witness indistinguishable proof system for the relation

$$\mathcal{R}_{\mathsf{NIWI}} := \left\{ \begin{matrix} (\mathsf{vk}, c, x), \\ w \end{matrix} \middle| (x, w) \in \mathcal{R} \quad \text{OR} \quad \begin{pmatrix} w = (u, r) \\ \mathsf{CLOWF}(\mathbb{1}^\lambda, u, \mathsf{vk}) = 1 \\ c = \mathsf{Com}\big(\mathbb{1}^\lambda, u;\, r\big) \end{pmatrix} \right\}$$

Finally, let $\mathsf{SOTS} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ be a strong one-time signature.

Theorem 24. *Let $a := (a_\lambda)_{\lambda \in \mathbb{N}}$ be a T-computable sequence. Assume the existence of a perfectly binding non-interactive commitment schemes and non-interactive witness-indistinguishable proofs without CRS. Furthermore, assume the existence of strong one-time signatures and subexponentially secure challengeless labelled one-way functions. Then, the construction in Fig. 47 is an a-disclosed simulation-sound NIZK without CRS for $\mathcal{R}$ with security against uniform PPT adversaries.*

Proof. Completeness follows immediately from the completeness of NIWI.

Claim. The construction in Fig. 47 is an a-disclosed multi-theorem zero-knowledge U-NIZK for $\mathcal{R}$.

Proof of the claim. Let $M(\lambda)$ be a polynomial upper-bounding the number of tuples (x_i, w_i) queried by $\mathcal{A}$. Since $\mathcal{A}$ is PPT, we know that M exists. Let π_i denote the answer to the i-th query.

```
A simulation-sound NIZK without CRS
$\mathsf{Prove}(1\!\!1^\lambda, x, w)$
  1. $(\mathsf{vk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{SOTS.Gen}(1\!\!1^\lambda)$
  2. $c \xleftarrow{\$} \mathsf{Com}(1\!\!1^\lambda, 0)$
  3. $\pi' \xleftarrow{\$} \mathsf{NIWI.Prove}\big(1\!\!1^\lambda, (\mathsf{vk}, c, x), w\big)$
  4. $s \leftarrow \mathsf{SOTS.Sign}\big(\mathsf{sk}, (\mathsf{vk}, c, \pi')\big)$
  5. Output $\pi := (\mathsf{vk}, c, \pi', s)$

$\mathsf{Verify}\big(\pi = (\mathsf{vk}, c, \pi', s), x\big)$
  1. $b_0 \leftarrow \mathsf{SOTS.Verify}\big(\mathsf{vk}, (\mathsf{vk}, c, \pi'), s\big)$
  2. $b_1 \leftarrow \mathsf{NIWI.Verify}\big(\pi', (\mathsf{vk}, c, x)\big)$
  3. Output $b_0 \wedge b_1$.

$\mathsf{SimSetup}(1\!\!1^\lambda)$
  1. Get the non-uniform advice $u_\lambda$ where $\mathsf{CLOWF}(1\!\!1^\lambda, \mathsf{Derive}(1\!\!1^\lambda, u_\lambda, \mathsf{id}), \mathsf{id}) = 1$
     for every $\mathsf{id}$.
  2. Output the empty string along with $\tau = u_\lambda$.

$\mathsf{SimProve}(\tau, x)$
  1. $(\mathsf{vk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{SOTS.Gen}(1\!\!1^\lambda)$
  2. $r \xleftarrow{\$} \{0,1\}^{q(\lambda)}$
  3. $u^{\mathsf{vk}} \xleftarrow{\$} \mathsf{Derive}(1\!\!1^\lambda, \tau, \mathsf{vk})$
  4. $c \leftarrow \mathsf{CCACom}(1\!\!1^\lambda, u^{\mathsf{vk}};\, r)$
  5. $\pi' \xleftarrow{\$} \mathsf{NIWI.Prove}\big(1\!\!1^\lambda, (\mathsf{vk}, c, x), (u^{\mathsf{vk}}, r)\big)$
  6. $s \leftarrow \mathsf{SOTS.Sign}\big(\mathsf{sk}, (\mathsf{vk}, c, \pi')\big)$
  7. Output $\pi := (\mathsf{vk}, c, \pi', s)$
```

Fig. 47. A simulation-sound NIZK without CRS

For every $i \in [M] \cup \{0\}$, we define Hybrid i to be the hybrid in which, for every $j \leq i$, we generate π_j using $\mathsf{SimProve}(\tau, x_j)$. For very $j > i$ instead, we generate π_j using $\mathsf{Prove}(1\!\!1^\lambda, x_j, w_j)$.

Notice that Hybrid 0 is identical to the zero-knowledge game when $b = 0$. Hybrid M is instead identical to the zero-knowledge game when $b = 1$. In order to prove our claim, it is sufficient to prove that no uniform PPT adversary can distinguish between Hybrid $i-1$ and Hybrid i for a randomly sampled $i \xleftarrow{\$} [M]$.

We show this by relying on a sequence of indistinguishable subhybrids.

Hybrid' 0. This hybrid coincides with Hybrid $i - 1$ for $i \xleftarrow{\$} [M]$.

Hybrid' 1. In this hybrid, we change the proof π_i. In particular, instead of committing to 0, we commit to the value $u^{\mathsf{vk}} \xleftarrow{\$} \mathsf{Derive}(1\!\!1^\lambda, \tau, \mathsf{vk})$ where τ is output by $\mathsf{SimSetup}$. All the rest remains as in Hybrid' 0.

This hybrid is indistinguishable from the previous one by the non-uniform hiding property of Com. In the reduction, the adversary $\mathcal{B}$ starts its execution by retrieving $\tau = u_\lambda$ and a_λ as part as its non-uniform advice. Then, it samples $i \xleftarrow{\$} [M]$ and runs $\mathcal{A}$. It replies to the first $i-1$ queries using $\mathsf{SimProve}$. Starting from the $(i+1)$-th query, it instead uses Prove.

The adversary $\mathcal{B}$ deviates from the game in the i-th Prove query. It begins by generating the signature key pair $(\mathsf{vk}, \mathsf{sk})$ and queries $(0, u^{\mathsf{vk}})$ to its challenger where $u^{\mathsf{vk}} \xleftarrow{\$} \mathsf{Derive}(\mathbb{1}^\lambda, \tau, \mathsf{vk})$. It then uses the answer c and $(\mathsf{vk}, \mathsf{sk})$ to generate π_i. The adversary $\mathcal{B}$ terminates the execution outputting the same value that $\mathcal{A}$ generates. Notice that if $\mathcal{A}$ distinguishes between Hybrid' 0 and Hybrid' 1, then $\mathcal{B}$ breaks the hiding properties of Com.

Hybrid' 2. In this hybrid, we change again the proof π_i. In particular, instead of using w_i as witness for NIWI, we use u^{vk} and the randomness used to commit to it. This hybrid is indistinguishable from the previous one by the S-witness indistinguishability of NIWI.

Notice that NIWI is secure against non-uniform adversaries so, in the reduction, we can assume that the adversary is given $\tau = u_\lambda$ and a_λ as part of its advice string. We build a PPT adversary that breaks the witness indistinguishability of NIWI. It starts its execution sampling a random $i \xleftarrow{\$} [M]$ and it simulates the game as in Hybrid' 1 to an internal copy of $\mathcal{A}$. It deviates from the game at the i-th Prove query. In particular, after generating vk and c as usual, it queries (vk, c, x_i) along with the witnesses w_i and (u^{vk}, r) to the NIWI challenger. Here, r denotes the randomness used for the generation of c. The answer π' is included in the proof returned to $\mathcal{A}$. At the end of its execution, $\mathcal{B}$ outputs the same bit as $\mathcal{A}$. Notice that if $\mathcal{A}$ succeeds in distinguishing, $\mathcal{B}$ succeeds too.

Observe that Hybrid' 2 is identical to Hybrid i for a random $i \xleftarrow{\$} [M]$. This ends the proof of the claim. ■

Claim. The construction in Fig. 47 is an a-disclosed simulation-sound U-NIZK for $\mathcal{R}$.

Proof of the claim. Suppose that there exists a uniform PPT adversary $\mathcal{A}$ that can generate an accepting proof for a false with non-negligible probability. We proceed by means of a series of hybrids.

Hybrid 0. In this hybrid, we provide the adversary with oracle access to $\mathsf{SimProve}$. The challenger outputs 1 when when the adversary terminates its execution outputting an accepting proof. Let Q denote the set of the oracle responses.

Hybrid 1. In this hybrid, when the adversary generates an accepting proof $\pi = (\mathsf{vk}, c, \pi', s)$ for a false statement x, the challenger checks whether vk coincides with the verification key in one of the responses of the simulation oracle. In that case, the challenger outputs 0, in the other cases it behaves as before.

Observe that Hybrid 0 and Hybrid 1 are indistinguishable due to the security of the strong one-time signature. Indeed, if the adversary can distinguish only if it generates an accepting proof $\pi = (\mathsf{vk}, c_0, c_1, \pi', s)$ for a false statement x such that $(\pi, x) \notin Q$ but vk was previously generated by $\mathsf{SimProve}$. In other words, $\mathcal{A}$ would produce a forgery with non-negligible probability. Notice that the one-time signature is secure against non-uniform adversaries, so in the reduction, we can assume that the adversary receives a_λ as part of its advice string.

Hybrid 2. In this hybrid, when the adversary generates an accepting proof $\pi = (\mathsf{vk}, c, \pi', s)$ for a false statement x, the challenger retrieves the value u hidden in c. If $\mathsf{CLOWF}(\mathbb{1}^\lambda, u, \mathsf{vk}) = 1$, the challenger outputs 0, in the other cases it behaves as before.

Hybrid 2 is indistinguishable from Hybrid 1 due to the $S(\lambda)$-security of the challengeless labelled one-way function. Indeed, if $\mathcal{A}$ distinguishes, it must be that with non-negligible probability, it outputs a proof $\widehat{\pi} = (\widehat{\mathsf{vk}}, \widehat{c}, \widehat{\pi}', \widehat{s})$ such that $\widehat{\mathsf{vk}}$ is different from all the signatures keys generates by the simulation oracle and $\widehat{c}$ is a commitment to a value $\widehat{u}$ such that $\mathsf{CLOWF}(\mathbb{1}^\lambda, \widehat{u}, \widehat{\mathsf{vk}}) = 1$. In the reduction, we build a uniform adversary $\mathcal{B}$ running in $\mathsf{poly}\big(\lambda, S(\lambda) + T(\lambda)\big)$ time. The adversary $\mathcal{B}$ simulates the game as in Hybrid 1 to an internal copy of $\mathcal{A}$. It starts its execution by retrieving a_λ in $\mathsf{poly}\big(\lambda, T(\lambda)\big)$ time. It replies to each simulation query of $\mathcal{A}$ by generating a signature key pair $(\mathsf{vk}, \mathsf{sk})$ and querying vk to its challenger. It uses the answer u^{vk} to generate the simulated proof. When $\mathcal{A}$ outputs a proof $\widehat{\pi} = (\widehat{\mathsf{vk}}, \widehat{c}, \widehat{\pi}', \widehat{s})$, $\mathcal{B}$ retrieves the value hidden in the corresponding commitment and outputs it along with $\widehat{\mathsf{vk}}$. The operation requires $\mathsf{poly}\big(\lambda, S(\lambda)\big)$ time. Notice that with non-negligible probability, the output is a pair $(\widehat{u}, \widehat{\mathsf{vk}})$ where $\mathsf{CLOWF}(\mathbb{1}^\lambda, \widehat{u}, \widehat{\mathsf{vk}}) = 1$ and $\widehat{\mathsf{vk}}$ has never been queries to the Derive oracle. We reached a contradiction.

Notice that by the perfect soundness of NIWI and the perfect binding property of Com, in Hybrid 2, the challenger always output 0 when the adversary outputs a proof for a false statement. Since Hybrid 0 and Hybrid 2 are indistinguishable, it must be that the adversary could not 0 generate valid proofs for false statements even in Hybrid 0. This terminates the proof. ■

□

9.3 Building simulation-extractable U-NIZKs without CRS

In this section, we will show how to build non-interactive arguments of knowledge without CRS with security against uniform adversaries. The idea is very simple: we use a non-interactive extractable commitment without CRS to hide the witness for our statement. Then, we use a (restricted) simulation-sound U-NIZK to prove the well-formedness of the commitment. In this way, we immediately obtain a (restricted) simulation-extractable U-NIZK without CRS. In order to extract the witness, we just need to extract the value hidden in the commitment using the corresponding trapdoor.

Non-interactive extractable commitments in the uniform setting. The main challenge of the approach described above is the fact that, in the non-uniform setting, non-interactive extractable commitments always need a CRS. If we aim for security against uniform adversaries only, we can however hope to built CRS-less constructions. In such schemes, the extraction trapdoor will be fixed but hard to retrieve for uniform adversaries.

In this section, we show how to build extractable commitments without CRS by relying on the uniform DDH and the uniform LWE assumptions. We start by formalising the definition.

Definition 45 (Non-interactive extractable U-commitment). *A non-interactive extractable U-commitment scheme is a pair of uniform PPT algorithms* $(\mathsf{ExCom}, \mathsf{Extract})$ *with the following syntax:*

- ExCom *is probabilistic and takes as input the security parameter* $\mathbb{1}^\lambda$ *and a message* m*. The output is a commitment* c*.*
- $\mathsf{Extract}$ *is deterministic, it takes as input a trapdoor* τ_e *and a commitment* c*. The output is a value* m *or* $\perp$*.*

We say that the scheme is perfectly correct if, for every $\lambda \in \mathbb{N}$*, there exits a value* τ_e *such that, for every message* m*,*

$$\Pr[\mathsf{Extract}(\tau_e, c) = m | c \xleftarrow{\$} \mathsf{ExCom}(\mathbb{1}^\lambda, m)] = 1.$$

We say that the scheme is perfectly binding if there exists no tuple (m_0, m_1, r_0, r_1) *such that* $m_0 \neq m_1$ *and* $\mathsf{ExCom}(\mathbb{1}^\lambda, m_0; r_0) = \mathsf{ExCom}(\mathbb{1}^\lambda, m_1; r_1)$*.*

For any function $S(\lambda)$*, we say that the scheme is* $S(\lambda)$*-uniformly hiding if no uniform adversary* $\mathcal{A}$ *running in* $\mathsf{poly}\big(\lambda, S(\lambda)\big)$ *time can distinguish between* $\mathsf{ExCom}(\mathbb{1}^\lambda, m_0)$ *and* $\mathsf{ExCom}(\mathbb{1}^\lambda, m_1)$ *even if it is the one choosing* m_0 *and* m_1*. We say that the scheme is post-quantumly* $S(\lambda)$*-uniformly hiding if even a uniform quantum adversary running in* $\mathsf{poly}\big(\lambda, S(\lambda)\big)$ *time cannot distinguish.*

Building non-interactive extractable commitments using uniform-DDH and uniform-LWE. The uniform-DDH assumption and the uniform-LWE assumptions immediately lead to non-interactive extractable U-commitments without CRS. In order to commit to a value m, it is sufficient to encrypt m either under the "ElGamal public-key" output by DDHGen or the "dual-Regev public-key" output by a trapdoored LWEGen. Extraction is a simple decryption using the trapdoors hidden in the DDHGen and LWEGen outputs.

We start by formalising the construction based on uniform-DDH.

Theorem 25. *Suppose that the* $S(\lambda)$*-uniform DDH assumption holds for* DDHGen*. Then, the construction in Fig. 48 is a non-interactive extractable bit-commitment scheme that is perfectly binding and* $S(\lambda)$*-uniformly hiding. Furthermore, it satisfies perfect correctness.*

Proof. Proving perfect correctness is straightforward. Even perfect binding is immediate as DDHGen always outputs a value $g \neq 1$.

Proving $S(\lambda)$-hiding is almost as simple. Suppose that there exists a uniform adversary $\mathcal{A}$ running in $\mathsf{poly}\big(\lambda, S(\lambda)\big)$ time that distinguishes between a commitment to 0 and one to 1. We proceed by a Hybrid argument.

```
NON-INTERACTIVE EXTRACTABLE BIT-COMMITMENT BASED ON UNIFORM-DDH
Let $a$ denote the value such that $h = g^a$ for $(\mathbb{G}, p, g, h) = \mathsf{DDHGen}(\mathbb{1}^\lambda)$
$\mathsf{ExCom}(\mathbb{1}^\lambda, b)$

  1. $(\mathbb{G}, p, g, h) \leftarrow \mathsf{DDHGen}(\mathbb{1}^\lambda)$
  2. $r \xleftarrow{\$} [p]$
  3. $c_0 \leftarrow g^r$
  4. $c_1 \leftarrow g^b \cdot h^r$
  5. Output $(c_0, c_1)$

$\mathsf{ExCom.Extract}(a, c_0, c_1)$

  1. Compute $m = c_1/c_0^a$. If $m = 1$ output 0, if $m = g$ output 1, otherwise, output
     $\perp$.
```

Fig. 48. Non-interactive extractable bit-commitment based on uniform-DDH

Hybrid 0. This hybrid corresponds to the usual hiding game for bit-commitments. In particular, the adversary is given a commitment to a random bit b.

Hybrid 1. In this hybrid, we provide the adversary with a pair (c_0, c_1) where $c_0 = g^r$ and $c_1 = g^b \cdot h^s$ for $r, s \xleftarrow{\$} [p]$. Notice that $\mathcal{A}$ cannot guess b with non-zero advantage as c_0 and c_1 are uniformly and independently distributed over $\mathbb{G}$. This contradicts the hardness of uniform DDH. Indeed, we can consider the adversary that, after receiving (g', h') from the uniform DDH challenger, samples a random bit b, sets $c_0 \leftarrow g'$ and $c_1 \leftarrow g^b \cdot h'$ and runs $\mathcal{A}$ on input (c_0, c_1). When $\mathcal{A}$ provides its answer b', the new adversary outputs 1 if and only if $b = b'$. If (g', h') were generated at random, the view of $\mathcal{A}$ is identical to the one in Hybrid 1. In the other case, the view is identical to the one in Hybrid 0. In the first case, the adversary outputs 1 with probability 1/2, in the second case, with probability $1/2 + \epsilon(\lambda)$ where $\epsilon(\lambda)$ is non-negligible. We reached a contradiction. $\square$

Below, we formalise the construction based on uniform-LWE.

Theorem 26. *Suppose that the post-quantum $S(\lambda)$-uniform LWE assumption holds for* LWEGen*. Assume also that* LWEGen *is trapdoored. Then, the construction in Fig. 49 is a post-quantum non-interactive, extractable bit-commitment scheme that is perfectly binding and $S(\lambda)$-uniformly hiding. Furthermore, it satisfies perfect correctness.*

Proof. Proving perfect correctness is straightforward. Indeed,

$$b' = u^\intercal \cdot c = u^\intercal \cdot A^\intercal s + u^\intercal \cdot x + b \cdot \lfloor q/2 \rfloor \cdot u^\intercal \cdot e_0 = u^\intercal \cdot x + b \cdot \lfloor q/2 \rfloor .$$

By Cauchy-Schwartz, $|u^\intercal \cdot x| \leq \|u\| \cdot \|x\| \leq \|u\| \cdot B(\lambda) < \lfloor q/4 \rfloor$. So, b' is always closer to $b \cdot \lfloor q/2 \rfloor$ than to $(1 - b) \cdot \lfloor q/2 \rfloor$.

Even perfect binding is immediate. Indeed, if there existed (s_0, x_0) and (s_1, x_1) such that $A^\intercal s_0 + x_0 = A^\intercal s_1 + x_1 + \lfloor q/2 \rfloor \cdot e_0$, then, we would have that

```
A Post-Quantum Non-interactive Extractable Bit-Commitment based on Uniform-LWE
Let $u \in \mathbb{Z}_2^m$ denote the value such that $u_0 = 1$, $\|u\| \cdot B(\lambda) < \lfloor q/4 \rfloor$ and $A \cdot u = 0$ for $A = \mathsf{LWEGen}(\mathbb{1}^\lambda)$. Let $e_0$ denote the first vector of the canocical basis of $\mathbb{Z}_q^m$.
$\mathsf{ExCom}(\mathbb{1}^\lambda, b)$

  1. $A \leftarrow \mathsf{LWEGen}(\mathbb{1}^\lambda)$
  2. $s \overset{\$}{\leftarrow} \mathbb{Z}_q^n$
  3. $x \overset{\$}{\leftarrow} \chi$
  4. $c \overset{\$}{\leftarrow} A^\intercal s + x + b \cdot \lfloor q/2 \rfloor \cdot e_0$
  5. Output $c$

$\mathsf{ExCom.Extract}(u, c)$

  1. Compute $b' = u^\intercal \cdot c$. If $b'$ is closer to $\lfloor q/2 \rfloor$ than to 0 modulo $q$, output 1,
     otherwise, output 1.
```

Fig. 49. Post-quantum non-interactive extractable bit-commitment based on uniform-LWE

$u^\intercal(x_0 - x_1) = \lfloor q/2 \rfloor$. However, by Cauchy-Schwartz, $|u^\intercal(x_0 - x_1)| \leq 2\|u\| \cdot B(\lambda) < 2\lfloor q/4 \rfloor \leq \lfloor q/2 \rfloor$.

Proving hiding is almost as simple. Suppose that there exists a quantum uniform adversary $\mathcal{A}$ running in $\mathsf{poly}\big(\lambda, S(\lambda)\big)$ time that distinguishes between a commitment to 0 and one to 1. We proceed by a Hybrid argument.

Hybrid 0. This hybrid corresponds to the usual hiding game for bit-commitments. In particular, the adversary is given a commitment to a random bit b.

Hybrid 1. In this hybrid, we sample b and we provide the adversary with a vector $c \overset{\$}{\leftarrow} \mathbb{Z}_q^m$. Notice that $\mathcal{A}$ cannot guess b with non-zero advantage as c is independent of it. This contradicts the hardness of quantum $S(\lambda)$-uniform LWE. Indeed, we can consider the adversary that, after receiving $v \in \mathbb{Z}_q^m$ from the uniform LWE challenger, samples a random bit b, sets $c \leftarrow v + b \cdot \lfloor q/2 \rfloor \cdot e_0$ and runs $\mathcal{A}$ on input c. When $\mathcal{A}$ provides its answer b', the new adversary outputs 1 if and only if $b = b'$. If v was generated at random, the view of $\mathcal{A}$ is identical to the one in Hybrid 1. In the other case, the view is identical to the one in Hybrid 0. In the first case, the adversary outputs 1 with probability $1/2$, in the second case, with probability $1/2 + \epsilon(\lambda)$ where $\epsilon(\lambda)$ is non-negligible. We reached a contradiction. □

Building (restricted) simulation-extractable U-NIZKs. We now formalise the idea described at the beginning of this section: using a simulation-sound U-NIZK and a non-interactive extractable U-commitment, we achieve simulation-extractability. If the U-NIZK is only restricted simulation-sound, the result will only achieve restricted simulation-extractability.

```
A simulation-extractable NIZK without CRS
Prove($1^\lambda, x, w$)

  1. $c \leftarrow \mathsf{ExCom}(1^\lambda, w;\, r)$
  2. $\pi' \xleftarrow{\$} \mathsf{NIZK}'.\mathsf{Prove}\big(1^\lambda, (c,x), (w,r)\big)$
  3. Output $\pi := (c, \pi')$

Verify$\big(\pi = (c,\pi'), x\big)$

  1. $b \leftarrow \mathsf{NIZK}'.\mathsf{Verify}\big(\mathsf{vk}, \pi', (c,x)\big)$
  2. Output $b$.

SimSetup($1^\lambda$)

  1. Get $\tau_s \leftarrow \mathsf{NIZK}'.\mathsf{SimSetup}(1^\lambda)$ and $\tau_e$
  2. Output the empty string along with $\tau = (\tau_s, \tau_e)$.

SimProve($\tau = (\tau_s, \tau_e), x$)

  1. $c \xleftarrow{\$} \mathsf{ExCom}(1^\lambda, 0)$
  2. $\pi' \xleftarrow{\$} \mathsf{NIZK}'.\mathsf{SimProve}\big(\tau_s, (c,x)\big)$
  3. Output $\pi := (c, \pi')$

Extract($\tau = (\tau_s, \tau_e), \pi = (c, \pi'), x$)

  1. $b \leftarrow \mathsf{Verify}(1^\lambda, \pi, x)$
  2. If $b = 0$, output $\bot$.
  3. $w \leftarrow \mathsf{ExCom}.\mathsf{Extract}(\tau_e, c)$
  4. If $(x, w) \in \mathcal{R}$, output $w$, otherwise, output $\bot$.
```

Fig. 50. A simulation-extractable NIZK without CRS

Let $T(\lambda)$ be a function of the security parameter and consider a T-computable sequence $a = (a_\lambda)_{\lambda \in \mathbb{N}}$. We make use of a non-interactive extractable U-commitment scheme ExCom without CRS satisfying perfect correctness. We require the scheme to be perfectly binding and $T(\lambda)$-uniformly hiding. We also rely on an a-compatible U-NIZK NIZK' for the relation

$$\mathcal{R}_{\mathsf{NIZK}'} := \left\{ \quad \big((c,x),(w,r)\big) \quad \middle| \begin{array}{l}(x,w) \in \mathcal{R} \\ c = \mathsf{ExCom}(1^\lambda, w;\, r)\end{array} \right\}$$

The U-NIZK NIZK' can either be restricted or fully simulation-sound. In the first case, we require that the running time of $\mathsf{NIZK}'.\mathsf{Check}$ is $\mathsf{poly}\big(\lambda, T(\lambda)\big)$. We also require that NIZK' is $T(\lambda)$-deterministic.

Theorem 27. *Assume the existence of subexponentially secure, perfectly correct non-interactive extractable U-commitments. If* NIZK' *is an a-disclosed (restricted) simulation-sound U-NIZK without CRS, then, the construction in Fig. 50 is an a-disclosed (restricted) simulation-extractable U-NIZK without CRS for* $\mathcal{R}$.

Proof. Completeness follows immediately from the completeness of NIWI.

Claim. The construction in Fig. 50 is an a-disclosed multi-theorem zero-knowledge U-NIZK for $\mathcal{R}$.

Proof of the claim. We proceed by means of a series of indistinguishable hybrids .

Hybrid 0. This hybrid coincides with the zero-knowledge game when $b = 0$. In particular, all the proof are generates using the witness.

Hybrid 1. In this hybrid, we modify each response of the oracle. In particular, instead of generating π' using $\mathsf{NIZK}'.\mathsf{Prove}$, we use $\mathsf{NIZK}'.\mathsf{SimProve}$.

Hybrid 1 is indistinguishable from Hybrid 0 under the a-disclosed multi-theorem zero-knowledge of NIZK'. In the reduction, we build a uniform PPT adversary $\mathcal{B}$ that simulates the game as in Hybrid 0 to an internal copy of $\mathcal{A}$. Observe that $\mathcal{B}$ receives a_λ from its challenger. The adversary $\mathcal{B}$ deviates from the game in each Prove query. In particular, after generating an extractable commitment c to the witness w using randomness r, $\mathcal{B}$ queries $\big((c, x), (w, r)\big)$ to its challenger and relays the answer along with c to $\mathcal{A}$. At the end of its execution, $\mathcal{B}$ outputs the same bit as $\mathcal{A}$. So, if $\mathcal{A}$ succeeds in distinguishing, $\mathcal{B}$ succeeds too.

Now, let $M(\lambda)$ be a polynomial upper-bounding the number of queries to Prove issued by the adversary. For every $i \in [M] \cup \{0\}$, we define the following hybrids.

Hybrid 2.i. In this hybrid, in the first i responses of the oracle, c will be a commitment to 0. In the remaining responses, c will instead be a commitment to the queried witness. The rest remains as in Hybrid 1. In particular, all the queries are simulated.

Observe that Hybrid 2.0 is identical to Hybrid 1. We show that Hybrid 2.i is indistinguishable from Hybrid 2.$(i-1)$ for a random choice of $i \xleftarrow{\$} [M]$ due to the hiding property of ExCom. In the reduction, we build a uniform adversary $\mathcal{B}$ running in $\mathsf{poly}\big(\lambda, T(\lambda)\big)$ time. The adversary $\mathcal{B}$ starts its execution by sampling $i \xleftarrow{\$} [M]$ and retrieving τ_s and a_λ. The operation requires $\mathsf{poly}\big(\lambda, T(\lambda)\big)$ time . Then, $\mathcal{B}$ simulates the game as in Hybrid 2.$(i-1)$ to an internal copy of $\mathcal{A}$ using τ_s. It deviates from the game at the i-th proof query. Specifically, after receiving the pair (x, w) from $\mathcal{A}$, it queries $(0, w)$ to its challenger. It then uses the answer c to generate the proof π requested by $\mathcal{A}$. At the end of its execution, $\mathcal{B}$ outputs the same bit as $\mathcal{A}$. So, if $\mathcal{A}$ succeeds in distinguishing, $\mathcal{B}$ does too. We reached a contradiction.

Observe that Hybrid 2.M is identical to the multi-theorem zero-knowledge game when $b = 1$. ■

Claim. If NIZK' is an a-disclosed (restricted) simulation-sound U-NIZK, then the construction in Fig. 50 is an a-disclosed (restricted) simulation-extractable U-NIZK for $\mathcal{R}$.

Proof of the claim. The first property is straightforward.

We focus on the second property starting from the case in which NIZK' is restricted simulation-sound. We use a hybrid argument.

Hybrid 0. This corresponds to the game in which all queries to the simulation oracle are answered using $\mathsf{SimProve}$. In particular, the responses are generated by committing to 0 and simulating π'. When the adversary outputs a pair (π, x) different from every oracle response, the challenger outputs $\mathsf{Verify}(\pi, x)$.

Now, let $M(\lambda)$ be a polynomial upper-bounding the number of queries to Prove issued by the adversary. For every $i \in [M] \cup \{0\}$, we define the following hybrids.

Hybrid 1.i In this hybrid, in the first i responses from the simulation oracle, we commit to the witness w queried by the adversary. In the remaining queries, we instead commit to 0. All the rest remains as in Hybrid 0.

Observe that Hybrid 1.0 is identical to Hybrid 0. We show that Hybrid 1.i and Hybrid 1.$(i-1)$ are indistinguishable for a random $i \xleftarrow{\$} [M]$ due to the hiding property of ExCom. In the reduction, we build a uniform adversary $\mathcal{B}$ running in $\mathsf{poly}\big(\lambda, T(\lambda)\big)$ time. The adversary $\mathcal{B}$ starts its execution by sampling $i \xleftarrow{\$} [M]$ and recovering τ_s and a_λ in $\mathsf{poly}\big(\lambda, T(\lambda)\big)$ time. Then, it simulates the game as in Hybrid 1.$(i-1)$ to an internal copy of $\mathcal{A}$. It deviates from the game at the i-th simulation query. Specifically, after receiving the witness w chosen by $\mathcal{A}$, $\mathcal{B}$ queries $(0, w)$ to its challenger. It uses the answer to generate the simulated proof π requested by $\mathcal{A}$. At the end of its execution $\mathcal{B}$ outputs the same bit as $\mathcal{A}$. So if $\mathcal{A}$ succeeds in distinguishing, $\mathcal{B}$ succeeds too.

Hybrid 2. In this hybrid, when the adversary provides a pair $(\pi = (c, \pi'), x)$, the challenger outputs $\mathsf{NIZK}'.\mathsf{Check}(\pi', (c, x))$. This hybrid is indistinguishable from Hybrid 1.M due to the a-disclosed restricted simulation-soundness of NIZK'. Suppose that there exists a uniform PPT adversary $\mathcal{A}$ that distinguishes between Hybrid 2 and Hybrid 1.M. Then, it must be that, with non-negligible probability, $\mathcal{A}$ can generate a pair $(\pi = (c, \pi'), x)$ different from every oracle response such that $\mathsf{Verify}(\pi, x) = \mathsf{NIZK}'.\mathsf{Verify}(\pi', (c, x)) \neq \mathsf{NIZK}'.\mathsf{Check}(\pi', (c, x))$. In the reduction, we build a uniform PPT adversary $\mathcal{B}$ that simulates the game as in Hybrid 1.M to an internal copy of $\mathcal{A}$. Notice that $\mathcal{B}$ receives a_λ from its challenger. The adversary $\mathcal{B}$ generates all responses to the simulation oracle by committing to the witness w queried by $\mathcal{A}$ using randomness r. Then, it queries (c, x) along with (w, r) to its own challenger. It relays the answer along with c to $\mathcal{A}$. When $\mathcal{A}$ provides a pair $(\pi = (c, \pi'), x)$, $\mathcal{B}$ outputs $\pi', (c, x)$. With non-negligible probability, $(\pi', (c, x))$ differs from all responses from the simulation oracle and $\mathsf{NIZK}'.\mathsf{Verify}(\pi', (c, x)) \neq \mathsf{NIZK}'.\mathsf{Check}(\pi', (c, x))$.

Hybrid 3.i In this hybrid, in the first i responses of the simulation oracle, we commit to 0. In the remaining queries, we instead commit to the witness w queried by the adversary. All the rest remains as in Hybrid 2.

Observe that Hybrid 3.0 is identical to Hybrid 2. We show that Hybrid 3.i and Hybrid 3.$(i-1)$ are indistinguishable for a random $i \xleftarrow{\$} [M]$ due to the hiding property of ExCom. In the reduction, we build a uniform adversary $\mathcal{B}$ running in $\mathsf{poly}\big(\lambda, T(\lambda)\big)$ time. The adversary $\mathcal{B}$ starts its execution by sampling

$i \xleftarrow{\$} [M]$ and recovering τ_s and a_λ in $\mathsf{poly}\big(\lambda, T(\lambda)\big)$ time. Then, it simulates the game as in Hybrid 3.$(i-1)$ to an internal copy of $\mathcal{A}$. It deviates from the game at the i-th simulation query. Specifically, after receiving the witness w chosen by $\mathcal{A}$, $\mathcal{B}$ queries $(w, 0)$ to its challenger. It uses the answer to generate the simulated proof π requested by $\mathcal{A}$. When $\mathcal{A}$ provides a pair $(\pi = (c, \pi'), x)$, $\mathcal{B}$ answers with $\mathsf{NIZK}'.\mathsf{Check}(\pi', (c, x))$. The operation requires $\mathsf{poly}\big(\lambda, T(\lambda)\big)$ time. At the end of its execution, $\mathcal{B}$ outputs the same bit as $\mathcal{A}$. So if $\mathcal{A}$ succeeds in distinguishing, $\mathcal{B}$ succeeds too.

Observe that we have proven that Hybrid 3.M is indistinguishable from Hybrid 0. That means that, even if we have oracle oracle access to $\mathsf{SimProve}$, it is hard to find a pair $(\pi = (c, \pi'), x)$ different from every oracle response such that $\mathsf{Verify}(\pi, x) \neq \mathsf{NIZK}'.\mathsf{Check}(\pi', (c, x))$. Notice that when $\mathsf{NIZK}'.\mathsf{Check}(\pi', (c, x)) = 1$, then $(c, x) \in L_{\mathsf{NIZK}'}$. By the perfect correctness of ExCom, we conclude that when $\mathsf{NIZK}'.\mathsf{Check}(\pi', (c, x)) = 1$, then extraction always succeeds. This terminates the proof for the case in which NIZK' is restricted simulation-sound.

We now focus on the case in which NIZK' is fully simulation-sound. Suppose that there exists an uniform PPT adversary $\mathcal{A}$ having oracle access to $\mathsf{SimProve}$ that, with non-negligible probability, can generate a proof that verifies but cannot be extracted. We can immediately derive a uniform PPT adversary $\mathcal{B}$ that breaks the simulation soundness of NIZK'. Such an adversary simulates the simulation-extractable game to an internal copy of $\mathcal{A}$. Notice that $\mathcal{B}$ obtains a_λ from its challenger. When $\mathcal{A}$ queries any value x to the simulation oracle, $\mathcal{B}$ generates an extractable commitment c to 0 and queries (c, x) to its challenger, it provides the answer along with c to $\mathcal{A}$. When $\mathcal{A}$ outputs a pair $\big(\pi = (c, \pi'), x\big)$, $\mathcal{B}$ outputs $\big(\pi', (c, x)\big)$. Observe that with non-negligible probability $\mathsf{Verify}\big(\pi', (c, x)\big) = 1$ but $\mathsf{Extract}(\tau_e, (c, \pi'), x) = \bot$. By the perfect correctness of the extractable commitment, it means that (c, x) does not belong to the language. In other words, $\mathcal{B}$ breaks the simulation soundness of NIZK'. This terminates our proof for the case in which NIZK' is fully simulation-sound.

■

□

10 Almost Everywhere Extractable NIZKs without CRS in the Uniform Setting

In this section, we show how to build almost everywhere extractable NIZKs without CRS in the uniform setting. Unfortunately, we were able to design only constructions that satisfy the standard definition of zero-knowledge (see Def. 33), but not chosen-ID zero-knowledge (see Def. 18). On the other hand, we achieved a stronger notion of almost everywhere extractability, which we called *simulation almost everywhere extractability.* This will be sufficient to build distributed samplers.

The main difference between the new notion and Def. 17 is that now, we require that, for every identity id, it is hard to find elements in $\mathsf{VPFE}_{\mathsf{id}}$ even if we have oracle access to the simulation oracle $\mathsf{SimProve}(\tau_s, \cdot, \cdot)$. Clearly, the adversary is not allowed to query for a simulated proof under the identity id. We recall that in the game the adversary is allowed to run in $\mathsf{poly}\big(\lambda, d(\lambda)\big)$ time where $d(\lambda)$ denotes an upper-bound on $|\mathsf{VPFE}_{\mathsf{id}}|$.

Definition 46 (Simulation almost everywhere extractable U-NIZK). *Let $a := (a_\lambda)_{\lambda\in\mathbb{N}}$ be a sequence of values. An identity-based U-NIZK* (Setup, $\mathsf{Prove}, \mathsf{Verify}$) *for $\mathcal{R}$ is a-compatible, simulation almost everywhere extractable if there exists a non-uniform PPT algorithm* $\mathsf{SimSetup}$ *and uniform PPT algorithms* $\mathsf{SimProve}$*,* Trap *and* $\mathsf{Extract}$ *such that*

1. *No non-uniform PPT adversary can distinguish between*

$$\Big\{\sigma\Big|\sigma \xleftarrow{\$} \mathsf{Setup}(\mathbb{1}^\lambda)\Big\} \qquad \Big\{\sigma\Big|(\sigma, \tau_s, \tau_e) \xleftarrow{\$} \mathsf{SimSetup}(\mathbb{1}^\lambda)\Big\}$$

2. *The algorithm* $\mathsf{Extract}$ *is deterministic and, for every $w = \mathsf{Extract}(\tau_e^{\mathsf{id}}, \pi, x)$,*

$$\Pr\Big[(x, w) \in \mathcal{R}\Big|w \neq \bot\Big] = 1.$$

3. *There exist efficiently computable values $\ell(\lambda) \in [m]$ and $d(\lambda)$ (the latter potentially superpolynomial) and a negligible function* $\mathsf{negl}(\lambda)$ *such that, for every identity* id*,*

$$\Pr\Big[\big|\mathsf{VPFE}_{\sigma,\tau_e,\mathsf{id}}\big| \leq d(\lambda)\Big|(\sigma, \tau_s, \tau_e) \xleftarrow{\$} \mathsf{NIZK.SimSetup}(\mathbb{1}^\lambda)\Big] \geq 1 - \mathsf{negl}(\lambda),$$

where

$$\mathsf{VPFE}_{\sigma,\tau_e,\mathsf{id}} := \left\{ \mathsf{Trunc}_\ell(\pi) \,\middle|\, \exists(x, r) \;\; s.t. \begin{array}{r} \mathsf{NIZK.Verify}(\sigma, \mathsf{id}, \pi, x) = 1 \\ \mathsf{NIZK.Trap}(\tau_e, \mathsf{id};\, r) = \tau_e^{\mathsf{id}} \\ \mathsf{NIZK.Extract}(\tau_e^{\mathsf{id}}, \pi, x) = \bot \end{array} \right\}$$

4. *Every uniform adversary $\mathcal{A}$ running in* $\mathsf{poly}(\lambda, d(\lambda))$ *time wins the game in Fig. 51 with negligible probability.*

The definition of zero-knowledge for simulation almost everywhere extractable U-NIZKs is formalised as in Def. 33 with minor changes to the notation. Indeed, since we are dealing with identity-based NIZKs, we need to augment the proof queries with identities id. The latter will be given as input to both Prove and $\mathsf{SimProve}$.

We now show that Lemma 2 can be generalised to simulation almost everywhere extractable U-NIZKs. In other words, we show that, if we deal with a subexponentially secure indistinguishability obfuscator and we rely on a simulation almost everywhere extractable U-NIZK, no *uniform* PPT adversary can distinguish between the obfuscation of C_0 and C_1 despite the existence of differing inputs. Furthermore, indistinguishability holds even if we provide the adversary with oracle access to $\mathsf{SimProve}(\tau_s, \cdot, \cdot)$ and we leak the extraction trapdoor τ_e. We recall that the circuits C_0 and C_1 were informally defined in Section 4.

Simulation Almost Everywhere Extractable U-NIZK Game

Initialisation: This procedure is run only once, at the beginning of the game.

1. $(\sigma, \tau_s, \tau_e) \xleftarrow{\$} \mathsf{SimSetup}(1\!\!1^\lambda)$
2. $Q \leftarrow \emptyset$
3. Activate $\mathcal{A}$ with $1\!\!1^\lambda$, $1\!\!1^{d(\lambda)}$, σ, τ_e and a_λ.

Prove: This procedure can be queried multiple times. Upon receiving any query $(\mathsf{Prove}, \mathsf{id}, x)$, compute the following.

1. Add id to Q.
2. $\pi \xleftarrow{\$} \mathsf{SimProve}(\tau_s, \mathsf{id}, x)$
3. Provide π to the adversary.

Win: The adversary wins if it outputs a pair (id, y) such that $\mathsf{id} \notin Q$ and $y \in \mathsf{VPFE}_{\sigma,\tau_e,\mathsf{id}}$.

Fig. 51. Simulation almost everywhere extractable U-NIZK game

Both the circuits take as input m statements and m NIZKs proving the validity of the latter. While C_0 simply verifies the NIZKs and outputs a function of the statements, C_1 tries to extract all the witnesses. In case of a failure, C_1 outputs $\perp$, otherwise, it performs the same operations as C_0.

Lemma 3. *Let $a := (a_\lambda)_{\lambda \in \mathbb{N}}$ be a sequence of values. Let NIZK be an a-disclosed, simulation almost everywhere extractable U-NIZK for the relation $\mathcal{R}$. Let $d(\lambda)$ be the upper-bound on $\left|\mathsf{VPFE}_{\sigma,\tau_e,\mathsf{id}}\right|$. Suppose that iO is an indistinguishability obfuscator against which every PPT adversary has advantage at most $\mathsf{negl}(\lambda)/d(\lambda)$. Then, no uniform PPT adversary $\mathcal{A}$ can win the game in Fig. 52 with non-negligible advantage.*

Proof. Let $\mathcal{A}$ be a uniform PPT adversary. We proceed by means of $m+1$ subhybrids indexed by $i = 0, 1, \ldots, m$. In the i-th of these hybrids, we provide $\mathcal{A}$ with an obfuscation of the program of C_i' (see Fig. 16) instead of one of C_0 or C_1. The rest remains as in Fig. 52.

Observe that by the security of iO, when $i = 0$, Hybrid i is indistinguishable from the game in Fig. 13 when $b = 0$. Similarly, by the security of iO, when $i = m$, Hybrid i is indistinguishable from from the game in Fig. 13 when $b = 1$. It remains to prove that $\mathcal{A}$ cannot distinguish between Hybrid $i - 1$ and Hybrid i for a random $i \xleftarrow{\$} [m]$. We rely on Lemma 1.

We consider the circuit sampler Samp_i that runs $\mathsf{SimSetup}$, provides σ, τ_e and a_λ to $\mathcal{A}$ and obtains $C, (\mathsf{id}_j)_{j\in[m]}$ after answering some queries to $\mathsf{SimProve}$. Then, it computes τ_e^j for every $j \in [m]$ and outputs C_{i-1}', C_i', $\rho := (\tau_s, \mathsf{id}_1, \ldots, \mathsf{id}_m)$ and aux corresponding to the internal state of $\mathcal{A}$ at the time it provides C and $(\mathsf{id}_j)_{j\in[m]}$. We consider the oracle $\mathcal{O}$ that, on input $\rho = (\tau_s, \mathsf{id}_1, \ldots, \mathsf{id}_m)$ and (id, x), it answers with $\mathsf{SimProve}(\tau_s, \mathsf{id}, x)$ as long as $\mathsf{id} \neq \mathsf{id}_j$ for every $j \in [m]$. We want to argue that even when aux is revealed and we give access to $\mathcal{O}(\rho, \cdot)$, diO Game for Simulation Almost Everywhere Extractable U-NIZKs

Initialisation: This procedure is run only once, at the beginning of the game.

1. $b \xleftarrow{\$} \{0,1\}$
2. $Q \leftarrow \emptyset$
3. $(\sigma, \tau_s, \tau_e) \xleftarrow{\$} \mathsf{SimSetup}(\mathbb{1}^\lambda)$
4. Activate the adversary $\mathcal{A}$ with $\mathbb{1}^\lambda$, σ, τ_e and a_λ.

Challenge: This query can be issued only once. On input a circuit C and identities $(\mathsf{id}_j)_{j\in[m]}$ such that $\mathsf{id}_j \notin Q$ for every $j \in [m]$, perform the following.

1. $\forall j \in [m]: \quad \tau_e^j \xleftarrow{\$} \mathsf{Trap}(\tau_e, \mathsf{id}_j)$
2. $\widetilde{C}_0 \xleftarrow{\$} \mathsf{iO}(\mathbb{1}^\lambda, C_0[\sigma, (\mathsf{id}_j)_{j\in[m]}])$ (see Fig. 14)
3. $\widetilde{C}_1 \xleftarrow{\$} \mathsf{iO}(\mathbb{1}^\lambda, C_1[\sigma, (\mathsf{id}_j)_{j\in[m]}, (\tau_e^j)_{j\in[m]}])$ (see Fig. 15)
4. Provide the adversary with $\widetilde{C}_b$

Prove: This procedure can be queried multiple times, both before and after choosing the challenge. On input any query $(\mathsf{Prove}, \mathsf{id}, x)$ where $\mathsf{id} \neq \mathsf{id}_j$ for every $j \in [m]$, provide the adversary with $\mathsf{SimProve}(\tau_s, \mathsf{id}, x)$ and add id to Q.

Win: The adversary wins if it guesses b.

Fig. 52. diO game for simulation almost everywhere extractable U-NIZKs

no uniform PPT adversary can distinguish between the obfuscation of C'_{i-1} and C'_i. That would immediately prove that $\mathcal{A}$ cannot distinguish between Hybrid $i-1$ and Hybrid i.

Let $\ell(\lambda)$ and $d(\lambda)$ be the values used in the third and fourth property of our simulation almost everywhere extractable U-NIZK. Let $\ell_0(\lambda)$ denote the position of the first bit of π_i. Define $\ell_1(\lambda) := \ell_0(\lambda) + \ell(\lambda)$.

The circuits C'_{i-1} and C'_i potentially have differing inputs. Observe that these must be values $(x_j, \pi_j)_{j\in[m]}$ for which $\mathsf{Verify}(\sigma, \mathsf{id}_i, \pi_i, x_i) = 1$ but $\mathsf{Extract}(\tau_e^i, \pi_i, x_i) = \perp$. In other words, we know that for every differing input,

$$\mathsf{DI}^{\ell_0,\ell_1}_{C'_{i-1},C'_i} \subseteq \mathsf{VPFE}_{\sigma,\tau_e,\mathsf{id}_i}.$$

With overwhelming probability over the randomness of $\mathsf{SimSetup}$, the latter has at most $d(\lambda)$ elements.

Now, suppose that there exists a uniform adversary $\mathcal{B}$ running in time $\mathsf{poly}\big(\lambda, d(\lambda)\big)$ that can find an element in $\mathsf{DI}^{\ell_0,\ell_1}_{C'_{i-1},C'_i}$ with non-negligible probability given C'_{i-1}, C'_i and aux and oracle access to $\mathcal{O}(\rho, \cdot)$. We build an adversary $\mathcal{B}'$ that breaks the fourth property of the simulation almost everywhere extractable U-NIZK.

The adversary $\mathcal{B}'$ runs an internal copy of $\mathcal{A}$ and one of $\mathcal{B}$. It starts by providing the NIZK CRS σ, the trapdoor τ_e and the value a_λ it received from its challenger to $\mathcal{A}$. It replies to the Prove queries of $\mathcal{A}$ by relaying the messages to its challenger. When $\mathcal{A}$ provides C and $(\mathsf{id}_j)_{j\in[m]}$, it computes $\tau_e^j \xleftarrow{\$} \mathsf{Trap}(\tau_e, \mathsf{id}_j)$

for every $j \in [m]$. Finally, it provides $\mathcal{B}$ with C'_{i-1}, C'_i and the internal state of $\mathcal{A}$. It answers the queries of $\mathcal{B}$ to $\mathcal{O}(\rho, \cdot)$ by relaying the messages to its challenger. Clearly, it does not reply when the queried identity coincides with id_j for any $j \in [m]$. The adversary $\mathcal{B}'$ terminates the execution outputting the same value as $\mathcal{B}$. We observe that $\mathcal{B}'$ outputs an element in $\mathsf{DI}^{\ell_0,\ell_1}_{C'_{i-1},C'_i}$ with non-negligible probability. Furthermore, it runs in uniform $\mathsf{poly}\big(\lambda, d(\lambda)\big)$ time. We reached a contradiction. So, the lemma follows from Lemma 1. □

10.1 Building simulation almost everywhere extractable U-NIZKs without CRS

We now present a simulation almost everywhere extractable U-NIZK without CRS. The construction is based on a perfectly sound NIWI, a challengeless labelled one-way function (see Def. 44) and two commitment schemes. The first one is perfectly binding and secure against non-uniform adversaries, the second one is an extractable U-commitment (see Def. 45).

Each proof includes two commitments c_0 and c_1. The first one hides the value 0, whereas the second one, which is extractable, hides the witness w. We use the NIWI to prove that either c_1 hides a witness for our statement, or c_0 is a commitment to a value u^{id} such that the pair $(u^{\mathsf{id}}, \mathsf{id})$ is accepted by the challengeless labelled one-way function. We recall that id is the identity under which we prove the validity of our statement. Observe that the soundness trapdoor coincides with the trapdoor of the labelled one-way function. The extraction trapdoor, instead, coincides with the trapdoor of the extractable commitment scheme.

Ensuring simulation almost everywhere extractability. Proving that our construction is zero-knowledge is rather straightforward using the techniques of Section 9. The main challenge is, however, proving simulation almost everywhere extractability.

As in Section 4.1, we would like that, in every valid NIZK where extraction fails, the commitment c_0 hides a value accepted by the labelled one-way function. In order to ensure this, we rely on a perfectly correct extractable U-commitment. In other words, we are sure that if c_1 hides the witness, the extraction always succeeds. By the perfect soundness of the NIWI, we are also sure that if c_1 does not hide the witness, then c_0 must hide a value accepted by the labelled one-way function in conjunction with id. If we consider a $B(\lambda)$-bounded labelled one-way function, the number of elements in $\mathsf{VPFE}_{\sigma,\tau_e,\mathsf{id}}$ is at most $d(\lambda) := B(\lambda) \cdot 2^{q(\lambda)}$ where $q(\lambda)$ denotes the length of the randomness needed by the perfectly binding commitment[17].

In order to achieve the last property of almost everywhere extractability, we require that the labelled OWF is at least $d(\lambda)$-secure. Observe that if we adopt a subexponentially secure injective, labelled OWF, by choosing a sufficiently large security parameter for the construction, we can always ensure this. We also

[17] As in Section 4.1, we can assume that $q(\lambda)$ is independent of the length of the committed value.

consider a perfectly binding commitment scheme in which the hidden message can be retrieved in $\mathsf{poly}\big(\lambda, d(\lambda)\big)$ time. In this way, we are sure that if an adversary can find an element in $\mathsf{VPFE}_{\sigma,\tau_e,\mathsf{id}}$ in $\mathsf{poly}\big(\lambda, d(\lambda)\big)$ time, it can also break the labelled one-way function. Notice that having access to simulated proofs under identities different from id does not help in finding elements in $\mathsf{VPFE}_{\sigma,\tau_e,\mathsf{id}}$.

Independently secure extractable commitments and labelled one-way functions. A minor issue we encounter in the blueprint above is that the proof of zero-knowledge would require that the extraction trapdoor is hard to retrieve even if we leak the soundness trapdoor. On the other hand, the proof of simulation almost everywhere extractability would require the symmetric relation: the soundness trapdoor is hard to retrieve even when the extraction trapdoor is known. Similarly to what we did in the first construction of simulation-sound U-NIZK, we therefore require that the security of the extractable U-commitment is "independent" of the security of the labelled one-way function.

Definition 47 (Independently secure extractable commitments and labelled OWFs). *Let* $(\mathsf{ExCom}, \mathsf{Extract})$ *be a non-interactive, extractable U-commitment and let* $(\mathsf{CLOWF}, \mathsf{Derive})$ *be a challengeless, labelled one-way function. Let* τ_e *be the trapdoor for* ExCom*. Similarly, let* τ_s *be the trapdoor for* CLOWF*. We say that* ExCom *and* CLOWF *are independently secure if*

- *For every pair of uniform PPT algorithms* $(\mathcal{A}_1, \mathcal{A}_2)$*,*

$$\Pr\left[\mathcal{A}_2(\psi, c) = b \,\middle|\, \begin{array}{l} b \xleftarrow{\$} \{0,1\} \\ (m_0, m_1, \psi) \xleftarrow{\$} \mathcal{A}_1(\mathbb{1}^\lambda, \tau_s) \\ c \xleftarrow{\$} \mathsf{ExCom}(\mathbb{1}^\lambda, m_b) \end{array}\right] = \mathsf{negl}(\lambda).$$

- *For every uniform PPT algorithm* $\mathcal{A}$*,*

$$\Pr\Big[\mathsf{id} \notin Q, \mathsf{CLOWF}(\mathbb{1}^\lambda, u^{\mathsf{id}}, \mathsf{id}) = 1 \,\Big|\, (\mathsf{id}, u^{\mathsf{id}}) \xleftarrow{\$} \mathcal{A}^{\mathsf{Derive}(\mathbb{1}^\lambda, \tau_s, \cdot)}(\mathbb{1}^\lambda, \tau_e)\Big] = \mathsf{negl}(\lambda),$$

  where Q *denotes the set of identities queried by* $\mathcal{A}$ *to* $\mathsf{Derive}(\mathbb{1}^\lambda, \tau_s, \cdot)$*.*

Let $S_1(\lambda), S_2(\lambda)$ *be functions of the security parameter. We say that* ExCom *and* CLOWF *are* (S_1, S_2)*-independently secure if the above properties hold even if* $\mathcal{A}$ *runs in* $\mathsf{poly}\big(\lambda, S_1(\lambda)\big)$ *time and* $\mathcal{A}_1$ *and* $\mathcal{A}_2$ *run in* $\mathsf{poly}\big(\lambda, S_2(\lambda)\big)$ *time.*

One way we can obtain pairs of independently secure challengeless labelled one-way functions and extractable U-commitments is by moving to a post-quantum world: by appropriately parametrising security, we can make τ_s harder to find than τ_e in a classical world. In a quantum world, instead, we can flip the relation: while finding τ_e will retain its hardness, τ_s will be easily retrievable. In other words, it is sufficient to take a post-quantumly secure non-interactive, extractable U-commitment and a post-quantumly broken challengeless labelled one-way function (or vice versa). This approach has clearly the disadvantage that the resulting pair is not secure against quantum adversaries, however, there may be other solutions that do not suffer from this issue.

Formalising the scheme. We rely on a perfectly binding non-interactive commitment scheme Com. We assume that Com requires $q(\lambda)$ bits of randomness. We also assume that the value hidden in a commitment can be retrieved with probability 1 in uniform $\mathsf{poly}\big(\lambda, 2^{q(\lambda)}\big)$ time.

Let $B(\lambda)$ and $T(\lambda)$ be functions of the security parameter such that $T(\lambda) \ll B(\lambda) \cdot 2^{q(\lambda)}$. We make use of a non-interactive extractable commitment scheme ExCom with perfect correctness (we use $q'(\lambda)$ to denote the length of the randomness it needs). We denote the extraction trapdoor by τ_e. We also use of an $B(\lambda)$-bounded challengeless labelled one-way function CLOWF. We denote the corresponding trapdoor by τ_s. We require that ExCom and CLOWF are $(T, 2^q \cdot B)$-independently secure.

Finally, let NIWI be a perfectly sound, witness indistinguishable proof system for the relation $\mathcal{R}_{\mathsf{NIWI}}$ described below

$$\left\{ \begin{matrix} \big((\mathsf{id}, c_0, c_1, x), \\ (w, r)\big) \end{matrix} \middle| \begin{pmatrix} (x, w) \in \mathcal{R} \\ c_1 = \mathsf{ExCom}(\mathbb{1}^\lambda, w;\, r) \end{pmatrix} \text{ OR } \begin{pmatrix} \mathsf{CLOWF}(\mathbb{1}^\lambda, w, \mathsf{id}) = 1 \\ c_0 = \mathsf{Com}\big(\mathbb{1}^\lambda, w;\, r\big) \end{pmatrix} \right\}$$

Theorem 28. *Let $q(\lambda)$, $T(\lambda)$ and $B(\lambda)$ be functions of the security parameter where $T(\lambda) \ll B(\lambda) \cdot 2^{q(\lambda)}$. Let a be any $T(\lambda)$-computable sequence. Assume the existence of non-interactive witness-indistinguishable proofs without CRS and a perfectly binding non-interactive commitment scheme. Suppose that the latter uses $q(\lambda)$ bits of randomness and it is possible to retrieve the committed values with probability 1 in uniform $\mathsf{poly}\big(\lambda, 2^{q(\lambda)}\big)$ time. Assume also the existence of a perfectly correct, non-interactive, extractable U-commitment and a $B(\lambda)$-bounded challengeless, labelled one-way function that are $(T, B \cdot 2^q)$-independently secure. Then, the construction in Fig. 53 is an a-disclosed multi-theorem zero-knowledge, simulation almost everywhere extractable U-NIZK without CRS for $\mathcal{R}$.*

Proof. Completeness follows immediately from the completeness of NIWI.

Claim. The construction in Fig. 53 is a-disclosed multi-theorem zero-knowledge against uniform PPT adversaries.

Proof of the claim. Let $M(\lambda)$ be a polynomial upper-bounding the number of tuples $(\mathsf{id}_i, x_i, w_i)$ queried by $\mathcal{A}$. Since $\mathcal{A}$ is PPT, we know that M exists. Let π_i denote the answer to the i-th query.

For every $i \in [M] \cup \{0\}$, we define Hybrid i as the hybrid in which, for every $j \leq i$, we generate π_j using $\mathsf{SimProve}(\tau_s, \mathsf{id}_j, x_j)$. For very $j > i$ instead, we generate π_j using $\mathsf{Prove}(\mathbb{1}^\lambda, \mathsf{id}_j, x_j, w_j)$.

Notice that Hybrid 0 is identical to the zero-knowledge game when $b = 0$. Hybrid M is instead identical to the zero-knowledge game when $b = 1$. In order to prove our claim, it is sufficient to show that no uniform PPT adversary can distinguish between Hybrid $(i-1)$ and Hybrid i for a randomly sampled $i \overset{\$}{\leftarrow} [M]$.

We rely on a sequence of indistinguishable subhybrids.

Hybrid' 0. This hybrid coincides with Hybrid $(i-1)$ for $i \overset{\$}{\leftarrow} [M]$.

```
A Simulation Almost Everywhere Extractable U-NIZK without CRS
$\mathsf{Prove}(\mathbb{1}^{\lambda}, \mathsf{id}, x, w)$

  1. $r \xleftarrow{\$} \{0,1\}^{q'(\lambda)}$
  2. $c_0 \xleftarrow{\$} \mathsf{Com}(\mathbb{1}^{\lambda}, 0)$
  3. $c_1 \leftarrow \mathsf{ExCom}(\mathbb{1}^{\lambda}, w; r)$
  4. $\pi' \xleftarrow{\$} \mathsf{NIWI.Prove}\big(\mathbb{1}^{\lambda}, (\mathsf{id}, c_0, c_1, x), (w, r)\big)$
  5. Output $\pi := (c_0, c_1, \pi')$

$\mathsf{Verify}\big(\mathsf{id}, \pi = (c_0, c_1, \pi'), x\big)$

  1. $b \leftarrow \mathsf{NIWI.Verify}\big(\pi', (\mathsf{id}, c_0, c_1, x)\big)$
  2. Output $b$.

$\mathsf{SimSetup}(\mathbb{1}^{\lambda})$

  1. Get the trapdoor $\tau_s$ for $\mathsf{CLOWF}$ and the extraction trapdoor $\tau_e$ for $\mathsf{ExCom}$.
  2. Output the empty string along with $\tau_s$ and $\tau_e$.

$\mathsf{SimProve}(\tau_s, \mathsf{id}, x)$

  1. $r \xleftarrow{\$} \{0,1\}^{q(\lambda)}$
  2. $u^{\mathsf{id}} \leftarrow \mathsf{Derive}(\tau_s, \mathsf{id})$
  3. $c_0 \leftarrow \mathsf{Com}(\mathbb{1}^{\lambda}, u^{\mathsf{id}}; r)$
  4. $c_1 \xleftarrow{\$} \mathsf{ExCom}(\mathbb{1}^{\lambda}, 0)$
  5. $\pi' \xleftarrow{\$} \mathsf{NIWI.Prove}\big(\mathbb{1}^{\lambda}, (\mathsf{id}, c_0, c_1, x), (u^{\mathsf{id}}, r)\big)$
  6. Output $\pi := (c_0, c_1, \pi')$

$\mathsf{Trap}(\tau_e, \mathsf{id})$

  1. Output $\tau_e^{\mathsf{id}} = (\tau_e, \mathsf{id})$

$\mathsf{Extract}(\tau_e^{\mathsf{id}} = (\tau_e, \mathsf{id}), \pi = (c_0, c_1, \pi'), x)$

  1. $b \leftarrow \mathsf{Verify}(\mathbb{1}^{\lambda}, \mathsf{id}, \pi, x)$
  2. If $b = 0$, output $\perp$.
  3. $w \leftarrow \mathsf{ExCom.Extract}(\tau_e, c_1)$
  4. If $(x, w) \in \mathcal{R}$, output $w$, otherwise, output $\perp$.
```

Fig. 53. A simulation almost everywhere extractable U-NIZK without CRS

Hybrid' 1. In this hybrid, we change the proof π_i. In particular, instead of generating a commitment c_0 of 0, we commit to $u^{\mathsf{id}_i} \xleftarrow{\$} \mathsf{Derive}(\tau_s, \mathsf{id}_i)$. All the rest remains as in Hybrid' 0.

This hybrid is indistinguishable from the previous one by the hiding property of Com. Since Com is secure against non-uniform adversaries, we can assume that τ_s and a_λ are given to the adversary $\mathcal{B}$ in the reduction as part of its advice string. So, the adversary $\mathcal{B}$ will simply sample $i \xleftarrow{\$} [M]$ and simulate the zero-knowledge game as in Hybrid' 0 to an internal copy of $\mathcal{A}$. At the i-th query, $\mathcal{B}$ will send the pair $(0, u^{\mathsf{id}_i})$ to its challenger and use the commitment it receives as part of the proof π_i. The adversary $\mathcal{B}$ terminates the execution outputting the same bit as $\mathcal{A}$. Notice that if $\mathcal{A}$ distinguishes between Hybrid' 0 and Hybrid' 1, then $\mathcal{B}$ break the hiding property of Com.

Hybrid' 2. In this hybrid, we change again the proof π_i. In particular, instead of using w_i as witness for NIWI, we use u^{id_i} and the randomness used to commit to it. This hybrid is indistinguishable from the previous one by the witness indistinguishability of NIWI.

Notice that NIWI is secure against non-uniform adversaries so, in the reduction, we can assume that the adversary is given τ_s and a_λ as part of its advice string. The adversary we construct, denoted by $\mathcal{B}$, starts its execution sampling a random $i \xleftarrow{\$} [M]$. Then, it simulates the game as in Hybrid' 1 to an internal copy of $\mathcal{A}$. It deviates from the game at the i-th Prove query. In particular, after generating c_0 and c_1 as usual, it queries $(\mathsf{id}_i, c_0, c_1, x_i)$ along with the witnesses (w_i, r_1) and (u^{id_i}, r_0) to the NIWI challenger. Here, r_0 and r_1 denote the randomness used for the generation of c_0 and c_1 respectively. The answer π' is included in the proof requested by $\mathcal{A}$. At the end of its execution, $\mathcal{B}$ outputs the same bit as $\mathcal{A}$. If $\mathcal{A}$ succeeds in distinguishing, $\mathcal{B}$ succeeds too.

Hybrid' 3. In this hybrid, in the i-th Prove query, instead of committing to w_i using ExCom, we commit to 0. This hybrid is indistinguishable from the previous one by the first property of $(T, B \cdot 2^q)$-independent security of ExCom and CLOWF.

In the reduction, we build a uniform adversary $\mathcal{B}$. The adversary $\mathcal{B}$ samples a random $i \xleftarrow{\$} [M]$ and retrieves τ_s and a_λ. The former is given by the challenger, the latter is computed in $T(\lambda)$-uniform polynomial time. Then, $\mathcal{B}$ simulates the game as in Hybrid' 2 to an internal copy of $\mathcal{A}$. In the i-th Prove query, $\mathcal{B}$ deviates from the protocol. In particular, it queries w_i and 0 to its challenger and uses the answer c_1 to generate the proof π_i. At the end of its execution, $\mathcal{B}$ outputs the same bit as $\mathcal{A}$. If $\mathcal{A}$ succeeds in distinguishing, $\mathcal{B}$ succeeds too.

Observe that Hybrid' 3 is identical to Hybrid i for a random $i \xleftarrow{\$} [M]$. This ends the proof of the claim. ■

Claim. The construction in Fig. 53 is an a-disclosed simulation almost everywhere extractable U-NIZK for $\mathcal{R}$.

Proof of the claim. The first two properties of simulation almost everywhere extractable NIZKs are trivial.

As for the third property, we observe that since NIWI is perfectly sound and ExCom perfectly correct, in all proofs $\pi = (c_0, c_1, \pi')$ for which there exists x such that $\mathsf{Verify}(\mathsf{id}, \pi, x) = 1$ but $\mathsf{Extract}(\tau_e^{\mathsf{id}}, \pi, x) = \bot$, c_0 is a commitment to a value u^{id} such that $\mathsf{CLOWF}(\mathbb{1}^\lambda, u^{\mathsf{id}}, \mathsf{id}) = 1$. The number of such commitments is at most $B(\lambda) \cdot 2^{q(\lambda)}$ where $q(\lambda)$ denotes the length of the randomness needed by Com. We conclude that the third property holds with relation to $\ell(\lambda)$ denoting the position of the last bit of c_0, $d(\lambda) := B(\lambda) \cdot 2^{q(\lambda)}$ and $\mathsf{VPFE}_{\sigma,\tau_e,\mathsf{id}} := \{\mathsf{Com}(\mathbb{1}^\lambda, u^{\mathsf{id}}; , r) | r \in \{0,1\}^{q(\lambda)}, \mathsf{CLOWF}(\mathbb{1}^\lambda, u^{\mathsf{id}}, \mathsf{id}) = 1\}$.

We focus on the last property. Suppose that there exists a uniform adversary $\mathcal{A}$ running in $\mathsf{poly}\big(\lambda, d(\lambda)\big)$ time that, even if knowing τ_e and having oracle access to $\mathsf{SimProve}$, it can derive a pair (id, y) where $y \in \mathsf{VPFE}_{\sigma,\tau_e,\mathsf{id}}$ and $\mathsf{id} \notin Q$ with non-negligible probability. We recall that Q denotes the set of identities queried to $\mathsf{SimProve}$.

We use $\mathcal{A}$ to build a uniform adversary $\mathcal{B}$ that breaks the second property of the $(T, B \cdot 2^q)$-independent security of ExCom and CLOWF. The adversary $\mathcal{B}$ runs in $\mathsf{poly}\big(\lambda, d(\lambda)\big)$ time. It starts its execution deriving τ_e and a_λ, the operation requires $T(\lambda)$ time (τ_e is given by the challenger). Then, it simulates the almost-everywhere extraction game as in Fig. 51 to an internal copy of $\mathcal{A}$. In every Prove query, $\mathcal{B}$ relays the corresponding identity id' to its challenger, it uses the answer $u^{\mathsf{id}'}$ to generate the simulated proof. When the adversary $\mathcal{A}$ outputs a pair (id, y) where $\mathsf{id} \notin Q$, $\mathcal{B}$ retrieves the value hidden in the commitment y and outputs it. Breaking y requires $\mathsf{poly}\big(\lambda, 2^{q(\lambda)}\big)$ time. Observe that $\mathcal{B}$ wins with non-negligible probability. This contradicts the security of challengeless labelled one-way function. ■

□

Acknowledgements

Damiano Abram thanks *Speedy's Tacos* for their delicious burritos and quesadillas. He also thanks the Aarhus Crypto Group and the people at NTT Research during summer 2022 for being amazing humans (independently of their success in research). The work of Damiano Abram was carried out during an internship funded by NTT Research.

References

ABB10. Shweta Agrawal, Dan Boneh, and Xavier Boyen. Efficient lattice (H)IBE in the standard model. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 553–572. Springer, Heidelberg, May / June 2010.

AJJM20. Prabhanjan Ananth, Abhishek Jain, Zhengzhong Jin, and Giulio Malavolta. Multi-key fully-homomorphic encryption in the plain model. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part I*, volume 12550 of *LNCS*, pages 28–57. Springer, Heidelberg, November 2020.

AOS23. Damiano Abram, Maciej Obremski, and Peter Scholl. On the (Im)possibility of Distributed Samplers: Lower Bounds and Party-Dynamic Constructions. Cryptology ePrint Archive, 2023/863, 2023.

ASY22. Damiano Abram, Peter Scholl, and Sophia Yakoubov. Distributed (correlation) samplers: How to remove a trusted dealer in one round. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part I*, volume 13275 of *LNCS*, pages 790–820. Springer, Heidelberg, May / June 2022.

BB04. Dan Boneh and Xavier Boyen. Secure identity based encryption without random oracles. In Matthew Franklin, editor, *CRYPTO 2004*, volume 3152 of *LNCS*, pages 443–459. Springer, Heidelberg, August 2004.

BCP14. Elette Boyle, Kai-Min Chung, and Rafael Pass. On extractability obfuscation. In Yehuda Lindell, editor, *TCC 2014*, volume 8349 of *LNCS*, pages 52–73. Springer, Heidelberg, February 2014.

BF01. Dan Boneh and Matthew K. Franklin. Identity-based encryption from the Weil pairing. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 213–229. Springer, Heidelberg, August 2001.

BGI$^+$01. Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 1–18. Springer, Heidelberg, August 2001.

BGI14. Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional signatures and pseudorandom functions. In Hugo Krawczyk, editor, *PKC 2014*, volume 8383 of *LNCS*, pages 501–519. Springer, Heidelberg, March 2014.

BL18a. Fabrice Benhamouda and Huijia Lin. k-round multiparty computation from k-round oblivious transfer via garbled interactive circuits. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 500–532. Springer, Heidelberg, April / May 2018.

BL18b. Nir Bitansky and Huijia Lin. One-message zero knowledge and non-malleable commitments. In Amos Beimel and Stefan Dziembowski, editors, *TCC 2018, Part I*, volume 11239 of *LNCS*, pages 209–234. Springer, Heidelberg, November 2018.

BOV03. Boaz Barak, Shien Jin Ong, and Salil P. Vadhan. Derandomization in cryptography. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 299–315. Springer, Heidelberg, August 2003.

BP04. Boaz Barak and Rafael Pass. On the possibility of one-message weak zero-knowledge. In Moni Naor, editor, *TCC 2004*, volume 2951 of *LNCS*, pages 121–132. Springer, Heidelberg, February 2004.

BP15. Nir Bitansky and Omer Paneth. ZAPs and non-interactive witness indistinguishability from indistinguishability obfuscation. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part II*, volume 9015 of *LNCS*, pages 401–427. Springer, Heidelberg, March 2015.

BSW16. Mihir Bellare, Igors Stepanovs, and Brent Waters. New negative results on differing-inputs obfuscation. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 792–821. Springer, Heidelberg, May 2016.

BW13. Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. In Kazue Sako and Palash Sarkar, editors, *ASIACRYPT 2013, Part II*, volume 8270 of *LNCS*, pages 280–300. Springer, Heidelberg, December 2013.

CCK$^+$22. Ran Canetti, Suvradip Chakraborty, Dakshita Khurana, Nishant Kumar, Oxana Poburinnaya, and Manoj Prabhakaran. COA-secure obfuscation and applications. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part I*, volume 13275 of *LNCS*, pages 731–758. Springer, Heidelberg, May / June 2022.

CHK03. Ran Canetti, Shai Halevi, and Jonathan Katz. A forward-secure public-key encryption scheme. In Eli Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 255–271. Springer, Heidelberg, May 2003.

CLTV15. Ran Canetti, Huijia Lin, Stefano Tessaro, and Vinod Vaikuntanathan. Obfuscation of probabilistic circuits and applications. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part II*, volume 9015 of *LNCS*, pages 468–497. Springer, Heidelberg, March 2015.

CM15. Michael Clear and Ciaran McGoldrick. Multi-identity and multi-key leveled FHE from learning with errors. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 630–656. Springer, Heidelberg, August 2015.

DHRW16. Yevgeniy Dodis, Shai Halevi, Ron D. Rothblum, and Daniel Wichs. Spooky encryption and its applications. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part III*, volume 9816 of *LNCS*, pages 93–122. Springer, Heidelberg, August 2016.

Gen06. Craig Gentry. Practical identity-based encryption without random oracles. In Serge Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 445–464. Springer, Heidelberg, May / June 2006.

GGH$^+$13. Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *54th FOCS*, pages 40–49. IEEE Computer Society Press, October 2013.

GGHW14. Sanjam Garg, Craig Gentry, Shai Halevi, and Daniel Wichs. On the implausibility of differing-inputs obfuscation and extractable witness encryption with auxiliary input. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part I*, volume 8616 of *LNCS*, pages 518–535. Springer, Heidelberg, August 2014.

GKLW21. Rachit Garg, Dakshita Khurana, George Lu, and Brent Waters. Black-box non-interactive non-malleable commitments. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part III*, volume 12698 of *LNCS*, pages 159–185. Springer, Heidelberg, October 2021.

GO07. Jens Groth and Rafail Ostrovsky. Cryptography in the multi-string model. In Alfred Menezes, editor, *CRYPTO 2007*, volume 4622 of *LNCS*, pages 323–341. Springer, Heidelberg, August 2007.

GOS06a. Jens Groth, Rafail Ostrovsky, and Amit Sahai. Non-interactive zaps and new techniques for NIZK. In Cynthia Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 97–111. Springer, Heidelberg, August 2006.

GOS06b. Jens Groth, Rafail Ostrovsky, and Amit Sahai. Perfect non-interactive zero knowledge for NP. In Serge Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 339–358. Springer, Heidelberg, May / June 2006.

HIJ$^+$17. Shai Halevi, Yuval Ishai, Abhishek Jain, Ilan Komargodski, Amit Sahai, and Eylon Yogev. Non-interactive multiparty computation without correlated randomness. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part III*, volume 10626 of *LNCS*, pages 181–211. Springer, Heidelberg, December 2017.

HV16. Carmit Hazay and Muthuramakrishnan Venkitasubramaniam. What security can we achieve within 4 rounds? In Vassilis Zikas and Roberto De

Prisco, editors, *SCN 16*, volume 9841 of *LNCS*, pages 486–505. Springer, Heidelberg, August / September 2016.

JLS21. Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from well-founded assumptions. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, page 60–73, New York, NY, USA, 2021. Association for Computing Machinery.

JLS22. Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from LPN over $\mathbb{F}_p$, DLIN, and PRGs in NC^0. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part I*, volume 13275 of *LNCS*, pages 670–699. Springer, Heidelberg, May / June 2022.

KK19. Yael Tauman Kalai and Dakshita Khurana. Non-interactive non-malleability from quantum supremacy. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 552–582. Springer, Heidelberg, August 2019.

KOR05. Jonathan Katz, Rafail Ostrovsky, and Michael O. Rabin. Identity-based zero knowledge. In Carlo Blundo and Stelvio Cimato, editors, *SCN 04*, volume 3352 of *LNCS*, pages 180–192. Springer, Heidelberg, September 2005.

KPTZ13. Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. Delegatable pseudorandom functions and applications. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013*, pages 669–684. ACM Press, November 2013.

KS17. Dakshita Khurana and Amit Sahai. How to achieve non-malleability in one or two rounds. In Chris Umans, editor, *58th FOCS*, pages 564–575. IEEE Computer Society Press, October 2017.

LPS17. Huijia Lin, Rafael Pass, and Pratik Soni. Two-round and non-interactive concurrent non-malleable commitments from time-lock puzzles. In Chris Umans, editor, *58th FOCS*, pages 576–587. IEEE Computer Society Press, October 2017.

LTV12. Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In Howard J. Karloff and Toniann Pitassi, editors, *44th ACM STOC*, pages 1219–1234. ACM Press, May 2012.

MW16. Pratyay Mukherjee and Daniel Wichs. Two round multiparty computation via multi-key FHE. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 735–763. Springer, Heidelberg, May 2016.

OSY21. Claudio Orlandi, Peter Scholl, and Sophia Yakoubov. The rise of paillier: Homomorphic secret sharing and public-key silent OT. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part I*, volume 12696 of *LNCS*, pages 678–708. Springer, Heidelberg, October 2021.

PVW08. Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. A framework for efficient and composable oblivious transfer. In David Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 554–571. Springer, Heidelberg, August 2008.

Sha84. Adi Shamir. Identity-based cryptosystems and signature schemes. In G. R. Blakley and David Chaum, editors, *CRYPTO'84*, volume 196 of *LNCS*, pages 47–53. Springer, Heidelberg, August 1984.

SW14. Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In David B. Shmoys, editor, *46th ACM STOC*, pages 475–484. ACM Press, May / June 2014.

Wat05. Brent R. Waters. Efficient identity-based encryption without random oracles. In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 114–127. Springer, Heidelberg, May 2005.

Zha16. Mark Zhandry. The magic of ELFs. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part I*, volume 9814 of *LNCS*, pages 479–508. Springer, Heidelberg, August 2016.

A Proof of Theorem 12

In this appendix, we prove Theorem 12.

Proof. Completeness is an immediate consequence of the completeness of NIWI.

Claim. The construction in Fig. 18 and Fig. 19 is an almost everywhere extractable NIZK.

Proof of the claim. We start by observing that the CRS σ generated by $\mathsf{Setup}(1\!\!1^\lambda)$ has exactly the same distribution as the one generated by $\mathsf{SimSetup}(1\!\!1^\lambda)$. The second property is also straightforward.

Therefore, we focus on the third property of the almost everywhere extractable NIZKs. Let $\ell(\lambda)$ denote the position of the last bit of c_0. We observe that by the perfect soundness of NIWI and the perfect correctness of IBE, in all proofs that verify but cannot be extracted, c_0 is a commitment to a preimage of v. Since the OWF is injective, there exists a unique preimage. Furthermore, the commitment algorithm takes as input $q_2(\lambda)$ bits of randomness. We conclude that, with probability 1, $\mathsf{VPFE}_{\sigma,\tau_e,\mathsf{id}}$ contains at most $2^{q_2(\lambda)}$ elements.

Now, suppose that an adversary $\mathcal{B}$ running in time $\mathsf{poly}\big(\lambda, 2^{q_2(\lambda)}\big)$ can find an element in $\mathsf{VPFE}_{\sigma,\tau_e,\mathsf{id}}$ with non-negligible probability after being provided with σ and τ_e. We can use this adversary to break the subexponential one-wayness of OWF. Indeed, consider the adversary that after receiving v from its challenger, generates $(\mathsf{mpk}, \mathsf{msk})$ using $\mathsf{IBE.Setup}$ and provides the pair $\sigma = (\mathsf{mpk}, v)$, $\tau_e = \mathsf{msk}$ to $\mathcal{B}$. When the latter replies with c_0, the new adversary retrieves the value hidden in c_0, breaking the hiding property of the commitment. The total running time of $\mathsf{poly}\big(\lambda, 2^{q_2(\lambda)}, S(\lambda)\big)$ and, with non-negligible probability, due to the perfectly binding property of the commitment scheme, the output is the value u such that $\mathsf{OWF}(1\!\!1^\lambda, u) = v$. We reached a contradiction. This ends the proof of the claim. ■

Claim. The construction in Fig. 18 and Fig. 19 satisfies chosen-ID zero-knowledge.

Proof of the claim. We prove the result by means of a sequence of indistinguishable hybrids. We repeat it for $i = 0, 1, \dots, M$ where M is a polynomial upper bound on the number of Prove queries issued by the adversary (since $\mathcal{A}$ is PPT, M exists). Throughout the proof, for any i, let π_i denote the proof provided to the adversary in the i-th Prove query. Let us denote the identity, the statement and the witness of the latter by id_i, x_i and w_i respectively.

Hybrid $i.0$. In this hybrid, for every $j \leq i$, we generate the proof π_j using $\mathsf{SimProve}(\tau_s, \mathsf{id}_j, x_j)$. For every $j > i$ instead, we generate the proof π_j using $\mathsf{Prove}(1\!\!1^\lambda, \sigma, \mathsf{id}_j, x_j, w_j)$. When $i = 0$, this hybrid corresponds to the execution of the chosen-ID zero-knowledge game (see Fig. 17) with $b = 0$. In particular, the proofs are all generated using the witnesses for $\mathcal{R}$. In all other cases, this hybrid is identical to Hybrid $(i-1).3$.

Hybrid $i.1$. In this hybrid, in the i-th proof, instead of committing to 0, we commit to $\tau_s = u$. This hybrid is indistinguishable from the previous one by the hiding property of the commitment scheme. Formally, the operations to generate π_i are the following. All other proofs are generated as in the previous hybrid.

1. $c_0 \xleftarrow{\$} \mathsf{Com}(1\!\!1^\lambda, u)$
2. $r \xleftarrow{\$} \{0,1\}^{q_1(\lambda)}$
3. $c_1 \leftarrow \mathsf{IBE.Enc}(\mathsf{mpk}, \mathsf{id}_i, w_i;\, r)$
4. $\pi' \xleftarrow{\$} \mathsf{NIWI.Prove}\big(1\!\!1^\lambda, (\mathsf{mpk}, v, \mathsf{id}_i, c_0, c_1, x_i), (w_i, r)\big)$
5. Output $\pi_i := (c_0, c_1, \pi')$

Hybrid $i.2$. In this hybrid, instead of using w_i and the randomness used to generate c_1 as witness for NIWI, we use u and the randomness used for c_0. By the witness indistinguishability of NIWI, this hybrid is indistinguishable from the previous one. Formally, the operations to generate π_i are the following. All other proofs are generated as in the previous hybrid.

1. $r \xleftarrow{\$} \{0,1\}^{q_2(\lambda)}$
2. $c_0 \leftarrow \mathsf{Com}(1\!\!1^\lambda, u;\, r)$
3. $c_1 \xleftarrow{\$} \mathsf{IBE.Enc}(\mathsf{mpk}, \mathsf{id}_i, w_i)$
4. $\pi' \xleftarrow{\$} \mathsf{NIWI.Prove}\big(1\!\!1^\lambda, (\mathsf{mpk}, v, \mathsf{id}_i, c_0, c_1, x_i), (u, r)\big)$
5. Output $\pi_i := (c_0, c_1, \pi')$

Hybrid $i.3$. In this hybrid, instead of encrypting w_i using IBE, we encrypt 0. This hybrid is indistinguishable from the previous one by the IND-ID-CPA security of IBE. Notice indeed, that we never provide the adversary with τ_e nor with $\tau_e^{\mathsf{id}_i}$. Formally, the operations to generate π_i are the following. All other proofs are generated as in the previous hybrid.

1. $r \xleftarrow{\$} \{0,1\}^{q_2(\lambda)}$
2. $c_0 \leftarrow \mathsf{Com}(1\!\!1^\lambda, u;\, r)$
3. $c_1 \xleftarrow{\$} \mathsf{IBE.Enc}(\mathsf{mpk}, \mathsf{id}_i, 0)$
4. $\pi' \xleftarrow{\$} \mathsf{NIWI.Prove}\big(1\!\!1^\lambda, (\mathsf{mpk}, v, \mathsf{id}_i, c_0, c_1, x_i), (u, r)\big)$
5. Output $\pi_i := (c_0, c_1, \pi')$

Notice that when $i = M$, the last hybrid is identical to the to the execution of the chosen-ID zero-knowledge game (see Fig. 17) with $b = 1$. This ends the proof of the claim. ■

□

B Proof of Theorem 16

In this appendix, we prove Theorem 16, namely that the construction in Fig. 34 is a lossy distributed sampler.

Proof. We observe that the second property of the lossy distributed sampler is a trivially implied by the ELF. Therefore, we focus on the first property, namely that for any polynomial $p(\lambda)$ and inverse polynomial function $\delta(\lambda)$, there exists a polynomial $q(\lambda)$ such that no adversary running in time at most $p(\lambda)$ can distinguish between the standard mode and the lossy mode parametrised by $q(\lambda)$ with advantage greater than $\delta(\lambda)$. We rely on an hybrid argument.

Hybrid 0. This hybrid corresponds to the game for lossy distributed samplers when the challenger uses the standard mode of operation.

We recap below the operations performed by $\mathsf{Setup}(1\!\!1^\lambda)$ in this hybrid.

1. $\sigma \xleftarrow{\$} \mathsf{NIZK.Setup}(1\!\!1^\lambda)$
2. $\sigma' \xleftarrow{\$} \mathsf{NIZK'.Setup}(1\!\!1^\lambda)$
3. Output (σ, σ').

The operations used by $\mathsf{Gen}\big(1\!\!1^\lambda, \mathsf{sid}, i, \mathsf{crs} = (\sigma, \sigma')\big)$ are instead the following.

1. $\rho_1 \xleftarrow{\$} \{0,1\}^{L_1(\lambda)}$
2. $\rho_2 \xleftarrow{\$} \{0,1\}^{L_2(\lambda)}$
3. $W \xleftarrow{\$} \{0,1\}^{\lambda}$
4. $(K_1^{(i)}, K_2^{(i)}, u_1, u_2) \leftarrow \mathsf{PRG}(W)$
5. $\mathsf{hk}_i \leftarrow \mathsf{Hash.Gen}(1\!\!1^\lambda;\, u_1)$
6. $\mathsf{EP}_i \leftarrow \mathsf{iO}(1\!\!1^\lambda, \mathsf{EProg}[K_1^{(i)}, K_2^{(i)}, i];\, u_2)$ (see Fig. 28)
7. $\mathsf{DP}_i \leftarrow \mathsf{iO}(1\!\!1^\lambda, \mathsf{DProg}[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma];\, \rho_1)$ (see Fig. 29)
8. $\pi_i \leftarrow \mathsf{NIZK.Prove}\big(\sigma, (\mathsf{sid}, i), (i, \mathsf{hk}_i, \mathsf{EP}_i), W;\, \rho_2\big)$
9. $\pi'_i \xleftarrow{\$} \mathsf{NIZK'.Prove}\big(\sigma', (\mathsf{sid}, i, \mathsf{hk}_i, \mathsf{EP}_i, \mathsf{DP}_i, \pi_i, \sigma), (W, \rho_1, \rho_2)\big)$
10. Output $U_i := (\mathsf{hk}_i, \mathsf{EP}_i, \mathsf{DP}_i, \pi_i, \pi_i)$.

Hybrid 1. In this hybrid, we simulate the NIZK CRS σ' and the proof π'_i sent in each $\mathsf{NewSession}$ query. We recall that i is the index queried by the adversary. Formally, the CRS of the distributed sample is now generate as follows.

1. $\sigma \xleftarrow{\$} \mathsf{NIZK.Setup}(1\!\!1^\lambda)$
2. $(\sigma', \tau') \xleftarrow{\$} \mathsf{NIZK'.SimSetup}(1\!\!1^\lambda)$
3. Output (σ, σ').

The operations performed by the challenger in order to compute U_i in $\mathsf{NewSession}$ queries become the following.

1. $W \xleftarrow{\$} \{0,1\}^{\lambda}$
2. $(K_1^{(i)}, K_2^{(i)}, u_1, u_2) \leftarrow \mathsf{PRG}(W)$
3. $\mathsf{hk}_i \leftarrow \mathsf{Hash.Gen}(1\!\!1^\lambda;\, u_1)$

4. $\mathsf{EP}_i \leftarrow \mathsf{iO}(1\!\!1^\lambda, \mathsf{EProg}[K_1^{(i)}, K_2^{(i)}, i]; u_2)$ (see Fig. 28)
5. $\mathsf{DP}_i \xleftarrow{\$} \mathsf{iO}(1\!\!1^\lambda, \mathsf{DProg}[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma])$ (see Fig. 29)
6. $\pi_i \xleftarrow{\$} \mathsf{NIZK.Prove}\big(\sigma, (\mathsf{sid}, i), (i, \mathsf{hk}_i, \mathsf{EP}_i), W\big)$
7. $\pi'_i \xleftarrow{\$} \mathsf{NIZK'.SimProve}\big(\tau', (i, \mathsf{sid}, \mathsf{hk}_i, \mathsf{EP}_i, \mathsf{DP}_i, \pi_i, \sigma)\big)$
8. Output $U_i := (\mathsf{hk}_i, \mathsf{EP}_i, \mathsf{DP}_i, \pi_i, \pi'_i)$.

This hybrid is indistinguishable from the previous one due to the multi-theorem zero-knowledge of $\mathsf{NIZK'}$. In the reduction, we build a PPT adversary $\mathcal{B}$ that simulates the lossy distributed sampler game as in Hybrid 0 to an internal copy of $\mathcal{A}$. The adversary $\mathcal{B}$ models the CRS using the element σ' obtained from its challenger. In each $\mathsf{NewSession}$ query, it generates the proof π'_i by querying the corresponding statement $(i, \mathsf{sid}, \mathsf{hk}_i, \mathsf{EP}_i, \mathsf{DP}_i, \pi_i, \sigma)$ and the relative witness (W, ρ_1, ρ_2) to its challenger. At the end of its execution, $\mathcal{B}$ outputs the same bit as $\mathcal{A}$. So if $\mathcal{A}$ succeeds at distinguishing between Hybrid 0 and Hybrid 1, $\mathcal{B}$ succeeds in breaking $\mathsf{NIZK'}$ too. Notice that if $\mathcal{A}$ is uniform $\mathcal{B}$ is uniform too.

Hybrid 2. In this hybrid, we change the reply to the sampling queries. In particular, for every U_l provided by the adversary in session sid, we compute

$$b_l \leftarrow \mathsf{NIZK'.Verify}\big(\sigma', \pi'_l, (l, \mathsf{sid}, \mathsf{hk}_l, \mathsf{EP}_l, \mathsf{DP}_l, \pi_l, \sigma)\big),$$
$$w_l \leftarrow \mathsf{NIZK'.Extract}\big(\tau', \pi'_l, (l, \mathsf{sid}, \mathsf{hk}_l, \mathsf{EP}_l, \mathsf{DP}_l, \pi_l, \sigma)\big).$$

If $b_l = 0$ or $w_l = \perp$, we provide the adversary with $\perp$. In all other cases, we provided it with $\mathsf{Sample}(U_1, \ldots, U_n, \mathsf{sid}, \mathsf{crs})$.

This hybrid is indistinguishable from the previous one due to the simulation-extractability of $\mathsf{NIZK'}$. Let $M(\lambda)$ denote a polynomial upper-bounding the number of sampling queries issued by the adversary. In the reduction, we build a PPT adversary $\mathcal{B}$ that simulates the lossy distributed sampler game as in Hybrid 0 to an internal copy of $\mathcal{A}$. The adversary $\mathcal{B}$ starts its execution by sampling $\iota \xleftarrow{\$} [M]$. It models the distributed sampler CRS using the element σ' obtained from its challenger. In each $\mathsf{NewSession}$ query, it generates the proof π'_i by querying the corresponding statement $(i, \mathsf{sid}, \mathsf{hk}_i, \mathsf{EP}_i, \mathsf{DP}_i, \pi_i, \sigma)$ to the simulation oracle. It replies to the first $\iota - 1$ sampling queries as in Hybrid 1. At the ι-th sampling query $\big(\mathsf{Sample}, \mathsf{sid}, (U_l)_{l \neq i}\big)$, however, $\mathcal{B}$ samples $j \xleftarrow{\$} [n] \setminus \{i\}$ and outputs $\pi'_j, (j, \mathsf{sid}, \mathsf{hk}_j, \mathsf{EP}_j, \mathsf{DP}_j, \pi_j, \sigma)$ where $\mathsf{hk}_j, \mathsf{EP}_j, \mathsf{DP}_j, \pi_j$ and π'_j are the elements in U_j.

If $\mathcal{A}$ succeeds at distinguishing, it must be that, with non-negligible probability $\epsilon(\lambda)$, one of its sampling queries contains a proof that verifies but cannot be extracted. With $1/M(\lambda)$ probability, the first proof of this kind will appear in the ι-th sampling query. So, $\mathcal{B}$ will succeed with probability at least $\epsilon(\lambda)/(M \cdot n)$. This contradicts the simulation-extractability of $\mathsf{NIZK'}$. Notice that if $\mathcal{A}$ is uniform $\mathcal{B}$ is uniform too.

Hybrid 3. In this hybrid, we simulate the NIZK CRS σ and the proof π_i sent in each $\mathsf{NewSession}$ query. We also modify the decryption program DP_i. Instead of verifying the NIZKs that are given as input, the program extracts the witness from them using trapdoors $(\tau_e^j)_{j \neq i}$ we hardcode into it. When extraction of any NIZK fails, the program outputs $\bot$. Each of the trapdoor is obtained as $\tau_e^j \xleftarrow{\$} \mathsf{NIZK.Trap}\big(\tau_e, (\mathsf{sid}, j)\big)$.

The distributed sampler CRS is now computed as follows.

1. $(\sigma, \tau_s, \tau_e) \xleftarrow{\$} \mathsf{NIZK.SimSetup}(1\!\!1^\lambda)$
2. $(\sigma', \tau') \xleftarrow{\$} \mathsf{NIZK'.SimSetup}(1\!\!1^\lambda)$
3. Output $\mathsf{crs} := (\sigma, \sigma')$

The operations performed by the challenger in order to compute U_i in $\mathsf{NewSession}$ queries become the following.

1. $W \xleftarrow{\$} \{0,1\}^\lambda$
2. $(K_1^{(i)}, K_2^{(i)}, u_1, u_2) \leftarrow \mathsf{PRG}(W)$
3. $\mathsf{hk}_i \leftarrow \mathsf{Hash.Gen}(1\!\!1^\lambda; u_1)$
4. $\mathsf{EP}_i \leftarrow \mathsf{iO}(1\!\!1^\lambda, \mathsf{EProg}[K_1^{(i)}, K_2^{(i)}, i]; u_2)$ (see Fig. 28)
5. $\forall j \neq i: \quad \tau_e^j \xleftarrow{\$} \mathsf{NIZK.Trap}\big(\tau_e, (\mathsf{sid}, j)\big)$
6. $\mathsf{DP}_i \xleftarrow{\$} \mathsf{iO}(1\!\!1^\lambda, \mathsf{DProg}_1[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j\neq i}])$ (see Fig. 32)
7. $\pi_i \xleftarrow{\$} \mathsf{NIZK.SimProve}\big(\tau_s, (\mathsf{sid}, i), (i, \mathsf{hk}_i, \mathsf{EP}_i)\big)$
8. $\pi_i' \xleftarrow{\$} \mathsf{NIZK'.SimProve}\big(\tau', (i, \mathsf{sid}, \mathsf{hk}_i, \mathsf{EP}_i, \mathsf{DP}_i, \pi_i, \sigma)\big)$
9. Output $U_i := (\mathsf{hk}_i, \mathsf{EP}_i, \mathsf{DP}_i, \pi_i, \pi_i')$.

Claim. If AClass denotes the class of non-uniform adversaries, Hybrid 3 is indistinguishable from Hybrid 2 due to the subexponential security of iO and almost-everywhere extractability and the chosen-ID zero-knowledge of NIZK.

Proof of the claim. We proceed by means of a sequence of indistinguishable hybrids.

Hybrid' 0. This hybrid corresponds to the game in Hybrid 2.

Hybrid' 1. In this hybrid, we generate the CRS σ using $\mathsf{SimSetup}$. In the process, we obtain also the trapdoors τ_e and τ_s. Hybrid' 1 and Hybrid' 0 are indistinguishable thanks to the first property of almost everywhere extractable NIZKs.

Let $M(\lambda)$ be a polynomial upper-bound on the number of $\mathsf{NewSession}$ queries issued by $\mathcal{A}$. We proceed with $M(\lambda)+1$ hybrids indexed by $\iota = 0, 1, \dots, M(\lambda)$.

Hybrid' 2.ι. In this hybrid, we reply to the first ι $\mathsf{NewSession}$ queries using an obfuscation of DProg_1 (see Fig. 32). Starting from the $(\iota+1)$-th query, we instead send an obfuscation of DProg (see Fig. 29). We observe that Hybrid' 2.0 is identical to Hybrid' 1. For every $\iota \in [M]$, Hybrid' 2.ι is indistinguishable from Hybrid' 2.$(\iota-1)$ thanks to Lemma 2. In the reduction, we build adversaries $\mathcal{B}_1$ and $\mathcal{B}_2$ that contradict Lemma 2. The adversary $\mathcal{B}_1$ receives σ and τ_e from the challenger. Then, it simulates the lossy distributed sampler game as in Hybrid' 2.$(\iota-1)$ to an internal copy of $\mathcal{A}$. At the ι-th $\mathsf{NewSession}$ query, $\mathcal{B}$ generates hk_i and EP_i as usual, then, it outputs the circuit $\mathsf{DProg}[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma]$ erasing the first two lines (i.e. the NIZKs verification) along with the identities $(\mathsf{sid}, l)_{l\neq i}$ and its internal state. The adversary $\mathcal{B}_2$ after receiving the obfuscated program DP_i and the internal state of $\mathcal{B}_1$ resumes the simulation of the lossy distributed sampler game with $\mathcal{A}$. It includes DP_i as part of the answer U_i to the ι-th $\mathsf{NewSession}$ query of $\mathcal{A}$. It produces the proofs π_i and π_i' in U_i as in Hybrid 2. At the end of its execution, $\mathcal{B}$ outputs the same bit as $\mathcal{A}$. Observe that if $\mathcal{A}$ distinguishes, then $(\mathcal{B}_1, \mathcal{B}_2)$ breaks Lemma 2.

Hybrid' 3. In this hybrid, we reply to all $\mathsf{NewSession}$ queries using an obfuscation of DProg_1 (see Fig. 32). Notice that Hybrid' 3 is identical to Hybrid' 2.M.

Hybrid' 4. In this hybrid, we simulate the proof π_i in each $\mathsf{NewSession}$ query. Hybrid' 3 and Hybrid' 4 are indistinguishable under the chosen-ID zero-knowledge of NIZK. In the reduction, we build a PPT adversary $\mathcal{B}$ that simulates the game as in Hybrid' 3 to an internal copy of $\mathcal{A}$. The NIZK CRS σ is given by the zero-knowledge challenger. In every execution of $\mathsf{NewSession}$, the adversary $\mathcal{B}$ generates $\mathsf{hk}_i, \mathsf{EP}_i, \mathsf{DP}_i, \pi_i'$ as in Hybrid' 3. The only difference is that it obtains the extraction trapdoors hardcoded into DP_i by querying $\big(\mathsf{Trap}, (\mathsf{sid}, j)\big)$ for every $j \neq i$ to the chosen-ID zero-knowledge challenger. The proof π_i is obtained by querying

$$\big(\mathsf{Prove}, (\mathsf{sid}, i), x := (i, \mathsf{hk}_i, \mathsf{EP}_i), w := W\big)$$

to the chosen-ID zero-knowledge challenger. Observe that $\mathcal{B}$ never queries proofs and trapdoors for the same identity. At the end of its execution $\mathcal{B}$ outputs the same bit as $\mathcal{A}$. So if $\mathcal{A}$ succeeds in distinguishing, $\mathcal{B}$ breaks the chosen-ID zero-knowledge property of NIZK.

Observe that Hybrid' 4 is identical to Hybrid 3. This terminates the proof of the claim. ■

Claim. If AClass denotes the class of uniform adversaries, Hybrid 3 is indistinguishable from Hybrid 2 due to the subexponential security of iO and the τ'-disclosed simulation almost-everywhere extractability and zero-knowledge of NIZK.

Proof of the claim. We proceed by means of a sequence of indistinguishable hybrids.

Hybrid' 0. This hybrid corresponds to the game in Hybrid 2.

Hybrid' 1. In this hybrid, we generate the CRS σ using $\mathsf{SimSetup}$. In the process, we obtain also the trapdoors τ_e and τ_s. Hybrid' 1 and Hybrid' 0 are indistinguishable thanks to the first property of simulation almost everywhere extractable NIZKs.

Hybrid' 2. In this hybrid, we simulate the proof π_i in each $\mathsf{NewSession}$ query. Hybrid' 1 and Hybrid' 2 are indistinguishable under zero-knowledge of NIZK. In the reduction, we build a PPT adversary $\mathcal{B}$ that simulates the game as in Hybrid' 1 to an internal copy of $\mathcal{A}$. The NIZK CRS σ and the value $a_\lambda = \tau'$ is given by the zero-knowledge challenger. In every execution of $\mathsf{NewSession}$, the adversary $\mathcal{B}$ generates $\mathsf{hk}_i, \mathsf{EP}_i, \mathsf{DP}_i, \pi_i'$ as in Hybrid' 1. The proof π_i is obtained by querying

$$\big(\mathsf{Prove}, (\mathsf{sid}, i), x := (i, \mathsf{hk}_i, \mathsf{EP}_i), w := W\big)$$

to the zero-knowledge challenger. At the end of its execution $\mathcal{B}$ outputs the same bit as $\mathcal{A}$. So if $\mathcal{A}$ succeeds in distinguishing, $\mathcal{B}$ breaks the zero-knowledge property of NIZK.

Let $M(\lambda)$ be a polynomial upper-bound on the number of $\mathsf{NewSession}$ queries issued by $\mathcal{A}$. We proceed with $M(\lambda) + 1$ hybrids indexed by $\iota = 0, 1, \ldots, M(\lambda)$.

Hybrid' 3.ι. In this hybrid, we reply to the first ι $\mathsf{NewSession}$ queries using an obfuscation of DProg_1 (see Fig. 32). Starting from the $(\iota + 1)$-th query, we instead send an obfuscation of DProg (see Fig. 29). We observe that Hybrid' 3.0 is identical to Hybrid' 2. We show that for a random $\iota \overset{\$}{\leftarrow} [M]$, Hybrid' 3.$\iota$ is indistinguishable from Hybrid' 3.$(\iota - 1)$ thanks to Lemma 3. In the reduction, we build an adversary $\mathcal{B}$ that contradicts Lemma 3. The adversary $\mathcal{B}$ receives σ, τ_e and $a_\lambda = \tau'$ from the challenger. Then, it simulates the lossy distributed sampler game as in Hybrid' 3.$(\iota - 1)$ to an internal copy of $\mathcal{A}$. In each $\mathsf{NewSession}$ query, $\mathcal{B}$ generates the simulated proof π_i by querying its challenger. At the ι-th $\mathsf{NewSession}$ query, $\mathcal{B}$ generates hk_i and EP_i as usual, then, it provides its challenger with the circuit $\mathsf{DProg}[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma]$ erasing the first two lines (i.e. the NIZKs verification) along with the identities $(\mathsf{sid}, l)_{l \neq i}$. Then, it includes the answer DP_i from its challenger as part of the distributed sampler message U_i. Notice that $\mathcal{B}$ never queries for a simulated proof with identity (sid, j) for any $j \neq i$. At the end of its execution, $\mathcal{B}$ outputs the same bit as $\mathcal{A}$. Observe that $\mathcal{B}$ is uniform so if $\mathcal{A}$ distinguishes, then $\mathcal{B}$ breaks Lemma 3.

Hybrid' 4. In this hybrid, we reply to all $\mathsf{NewSession}$ queries using an obfuscation of DProg_1 (see Fig. 32). Notice that Hybrid' 4 is identical to Hybrid' 3.M.

Observe that Hybrid' 4 is identical to Hybrid 3. This terminates the proof of the claim. ■

We now proceed by repeating the following sequence of hybrids for $\iota = 1, \ldots, M(\lambda), M(\lambda)+1$ where $M(\lambda)$ is a polynomial upper-bound on the number of $\mathsf{NewSession}$ queries issued by the adversary. From now on, all pairs of hybrids can be proven indistinguishable by means of reductions to primitives that are secure against non-uniform adversaries. For this reason, we do not need to worry anymore on how the reduction obtains τ_e and τ_s.

Hybrid 4.ι.0. In this hybrid, the challenger starts its execution by generating a ELF $f \overset{\$}{\leftarrow} \mathsf{ELF.Gen}(M, M)$. Notice that the latter is in injective mode. The input space is chosen sufficiently big to embed all tuples $(\mathsf{hk}_j, \mathsf{EP}_j)_{j \neq i}$ into it without collisions.

The challenger generates the answer U_i to the first $\iota - 1$ $\mathsf{NewSession}$ queries as follows.

1. $K_1^{(i)} \overset{\$}{\leftarrow} F_1.\mathsf{Gen}(1\!\!1^\lambda)$
2. $K_2^{(i)} \overset{\$}{\leftarrow} F_2.\mathsf{Gen}(1\!\!1^\lambda)$
3. $\mathsf{hk}_i \overset{\$}{\leftarrow} \mathsf{Hash.Gen}(1\!\!1^\lambda)$
4. $K \overset{\$}{\leftarrow} F.\mathsf{Gen}(1\!\!1^\lambda)$
5. $\mathsf{EP}_i \overset{\$}{\leftarrow} \mathsf{iO}(1\!\!1^\lambda, \mathsf{EProg}_{\mathsf{Ls}}[K_2^{(i)}, i])$ (see Fig. 30)

6. $\forall j \neq i: \quad \tau_e^j \xleftarrow{\$} \mathsf{NIZK.Trap}\big(\tau_e, (\mathsf{sid}, j)\big)$
7. $\mathsf{DP}_i \xleftarrow{\$} \mathsf{iO}(1\!\!1^\lambda, \mathsf{DProg}_{\mathsf{Ls}}[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j\neq i}, K, f])$ (see Fig. 31)
8. $\pi_i \xleftarrow{\$} \mathsf{NIZK.SimProve}\big(\tau_s, (\mathsf{sid}, i), (i, \mathsf{hk}_i, \mathsf{EP}_i)\big)$
9. $\pi_i' \xleftarrow{\$} \mathsf{NIZK'.SimProve}\big(\tau', (i, \mathsf{sid}, \mathsf{hk}_i, \mathsf{EP}_i, \mathsf{DP}_i, \pi_i, \sigma)\big)$
10. Output $U_i := (\mathsf{hk}_i, \mathsf{EP}_i, \mathsf{DP}_i, \pi_i, \pi_i')$.

All the remaining NewSession queries are answered as in Hybrid 3. Notice that when $\iota = 0$, Hybrid 4.ι.0 is identical to Hybrid 2. In all other cases, Hybrid 4.ι.0 is identical to Hybrid 4.$(\iota - 1)$.

Hybrid 4.ι.1. In this hybrid, we generate the pair $(\mathsf{hk}_i, \mathsf{EP}_i)$ in the ι-th NewSession query using full-entropy randomness instead of by expanding a PRG seed. All the rest remains as in the previous hybrid. This hybrid is indistinguishable from the previous one by the security of the PRG (the proof is an easy reduction). The operations performed by the challenger in order to compute U_i in the ι-th NewSession query become the following.

1. $K_1^{(i)} \xleftarrow{\$} F_1.\mathsf{Gen}(1\!\!1^\lambda)$
2. $K_2^{(i)} \xleftarrow{\$} F_2.\mathsf{Gen}(1\!\!1^\lambda)$
3. $\mathsf{hk}_i \xleftarrow{\$} \mathsf{Hash.Gen}(1\!\!1^\lambda)$
4. $\mathsf{EP}_i \xleftarrow{\$} \mathsf{iO}(1\!\!1^\lambda, \mathsf{EProg}[K_1^{(i)}, K_2^{(i)}, i])$ (see Fig. 28)
5. $\forall j \neq i: \quad \tau_e^j \xleftarrow{\$} \mathsf{NIZK.Trap}\big(\tau_e, (\mathsf{sid}, j)\big)$
6. $\mathsf{DP}_i \xleftarrow{\$} \mathsf{iO}(1\!\!1^\lambda, \mathsf{DProg}_1[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j\neq i}])$ (see Fig. 32)
7. $\pi_i \xleftarrow{\$} \mathsf{NIZK.SimProve}\big(\tau_s, (\mathsf{sid}, i), (i, \mathsf{hk}_i, \mathsf{EP}_i)\big)$
8. $\pi_i' \xleftarrow{\$} \mathsf{NIZK'.SimProve}\big(\tau', (i, \mathsf{sid}, \mathsf{hk}_i, \mathsf{EP}_i, \mathsf{DP}_i, \pi_i, \sigma)\big)$
9. Output $U_i := (\mathsf{hk}_i, \mathsf{EP}_i, \mathsf{DP}_i, \pi_i, \pi_i')$.

Remark 1. From now on, we will keep the generation of $K_1^{(i)}$, $K_2^{(i)}$ and hk_i implicit. Indeed, the procedure will remain as in Hybrid 4.ι.1. We do the same for π_i and π_i' and $(\tau_e^j)_{j\neq i}$.

Hybrid 4.ι.2. In this hybrid, we modify the answer to the sampling queries concerning the ι-th session. In particular, when the NIZKs verify and we succeed in extracting the witnesses $(W_j)_{j\neq i}$ from the messages $(U_j)_{j\neq i}$ provided by the adversary, we answer the query as follows.

1. $\forall j \in [n]: y_j \leftarrow \mathsf{Hash}\big(\mathsf{hk}_j, (\mathsf{hk}_l, \mathsf{EP}_l)_{l\neq j}\big)$
2. $\forall j \neq i: (K_1^{(j)}, K_2^{(j)}, u_1^j, u_2^j) \leftarrow \mathsf{PRG}(W_j)$
3. $\forall j \in [n]: s_j \leftarrow F_1(K_1^{(j)}, y_j)$
4. $R \leftarrow \mathcal{D}(1\!\!1^\lambda; s_1 \oplus \cdots \oplus s_n)$
5. Provide R to the adversary.

Observe that this hybrid is identical to the previous one by the perfect correctness of multi key FHE and the perfect correctness and injectivity of iO. The latter is needed to argue that EP_j univocally determines $K_1^{(j)}$ and $K_2^{(j)}$.

Hybrid 4.ι.3. In this hybrid, we change both the encryption program EP_i and the decryption program DP_i generated for the ι-th $\mathsf{NewSession}$ query, switching to an obfuscation of $\mathsf{EProg}_{\mathsf{Ls}}$ (see Fig. 30) and DProg_2 (see Fig. 33). All the rest remains as in the previous hybrid. Notice that, at this point, we removed $K_1^{(i)}$ from the code of EP_i. The operations performed by the challenger in order to compute U_i in the ι-th $\mathsf{NewSession}$ query become the following.

1. $\mathsf{EP}_i \xleftarrow{\$} \mathsf{iO}(\mathbb{1}^\lambda, \mathsf{EProg}_{\mathsf{Ls}}[K_2^{(i)}, i])$ (see Fig. 30)
2. $\mathsf{DP}_i \xleftarrow{\$} \mathsf{iO}(\mathbb{1}^\lambda, \mathsf{DProg}_2[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j \neq i}, K_1^{(i)}])$ (see Fig. 33)
3. Output $U_i := (\mathsf{hk}_i, \mathsf{EP}_i, \mathsf{DP}_i, \pi_i, \pi_i')$.

Claim. Assuming the subexponential security of iO, of Hash, of the puncturable PRF F_2 and of multi-key FHE, no PPT adversary can distinguish between Hybrid 4.ι.2 and Hybrid 4.ι.3.

Proof of the claim. We select the security parameter of the subexponentially collision resistance hash function so that, for any PPT adversary,

$$2^{2\lambda\cdot(n-1)} \cdot \mathsf{Adv}_{\mathsf{CR}}^{\mathcal{A}}(\lambda) = \mathsf{negl}(\lambda).$$

Let Ω be the set of all the tuples $(\mathsf{hk}_j, \mathsf{EP}_j)_{j \neq i}$ such that each $(\mathsf{hk}_j, \mathsf{EP}_j)$ is generated by expanding a λ-bit PRG seed as in Fig. 34. Observe that $|\Omega| \leq 2^{\lambda\cdot(n-1)}$. We conclude that with overwhelming probability over hk_i, there exist no collisions in Ω. Otherwise, the adversary that simply outputs two random elements in Ω would break the above assumption. We can therefore, prove indistinguishability conditioned on this event occurring.

We proceed once again by means of a series of hybrids. Their number will however be superpolynomial. Specifically, we consider the set of all possible digests $\{0,1\}^{t(\lambda)}$ and we set $\hat{y}$ to its minimum according to the lexicographical order. We proceed through the following sequence of hybrids gradually increasing $\hat{y}$ until it reaches the maximum in $\{0,1\}^{t(\lambda)}$.

Hybrid $\hat{y}$.0. In this hybrid, we modify the programs EP_i and DP_i sent in the ι-th $\mathsf{NewSession}$ query. In particular, instead of obfuscating EProg (see Fig. 28), we obfuscate EProg^0 (see Fig. 54). Similarly, instead of obfuscating DProg_1 (see Fig. 32), we obfuscate DProg_1^0 (see Fig. 55). In both these programs, we hardcode $\hat{y}$. If the digest input in EP_i is strictly lexicographically smaller than $\hat{y}$, the encryption program performs the same operations as $\mathsf{EProg}_{\mathsf{Ls}}$ (see Fig. 30), otherwise, it behaves as EProg (see Fig. 28). Similarly, if the hash y_i of the tuple $(\mathsf{hk}_j, \mathsf{EP}_j)_{j \neq i}$ input in DP_i is strictly lexicographically smaller than $\hat{y}$, the decryption program performs the same operations as DProg_2 (see Fig. 33), otherwise, it behave as DProg_1 (see Fig. 32).

Notice that when $\hat{y}$ is the minimum of $\{0,1\}^{t(\lambda)}$, the new programs have exactly the same input-output behaviour as EProg and DProg_1. We conclude that, when $\hat{y}$ is minimum, Hybrid $\hat{y}$.0 is indistinguishable from Hybrid 4.ι.1, by the security of iO. If $\hat{y}$ is not the minimum, this hybrid will be identical to the previous one (i.e. Hybrid $\hat{y}'$.7 where $\hat{y}'$ is the previous value of $\hat{y}$).

The operations performed by the challenger in order to compute U_i in the ι-th $\mathsf{NewSession}$ query become the following.

$\mathsf{EProg}^0[K_1^{(i)}, K_2^{(i)}, i, \hat{y}]$

Hard-coded. The PPRF keys $K_1^{(i)}$ and $K_2^{(i)}$, the index i, the hybrid index $\hat{y}$.
Input. A digest $y \in \{0,1\}^{t(\lambda)}$.

```
1. If $y <_{\mathsf{lex}} \hat{y}$: $(\mathsf{pk}_i, c_i) \leftarrow \mathsf{EProg}_{\mathsf{Ls}}[K_2^{(i)}, i](y)$ (see Fig. 30)
2. Otherwise, $(\mathsf{pk}_i, c_i) \leftarrow \mathsf{EProg}[K_1^{(i)}, K_2^{(i)}, i](y)$ (see Fig. 28)
3. Output $(\mathsf{pk}_i, c_i)$
```

Fig. 54. Hybrid $\hat{y}$.0: the unobfuscated encryption program of party P_i

$\mathsf{DProg}_1^0[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j \neq i}, K_1^{(i)}, \hat{y}]$

Hard-coded. The index i of the party, the session identity sid, a PPRF key $K_2^{(i)}$, the encryption program EP_i, the hash key hk_i, the extractable NIZK CRS σ and the extraction trapdoors $(\tau_e^j)_{j \neq i}$, the PPRF key $K_1^{(i)}$, the hybrid index $\hat{y}$.
Input. Set of $n-1$ tuples $(\mathsf{hk}_j, \mathsf{EP}_j, \pi_j)_{j \neq i}$.

```
1. If $\mathsf{Hash}\big(\mathsf{hk}_i, (\mathsf{hk}_j, \mathsf{EP}_j)_{j \neq i}\big) <_{\mathsf{lex}} \hat{y}$:
   $d_i \leftarrow \mathsf{DProg}_2[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j \neq i}, K_1^{(i)}]\Big((\mathsf{hk}_j, \mathsf{EP}_j, \pi_j)_{j \neq i}\Big)$ (see Fig. 33)
2. Otherwise,
   $d_i \leftarrow \mathsf{DProg}_1[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j \neq i}]\Big((\mathsf{hk}_j, \mathsf{EP}_j, \pi_j)_{j \neq i}\Big)$ (see Fig. 32)
3. Output $d_i$
```

Fig. 55. Hybrid $\hat{y}$.0: the unobfuscated decryption program of party P_i

1. $\mathsf{EP}_i \xleftarrow{\$} \mathsf{iO}(\mathbb{1}^\lambda, \mathsf{EProg}^0[K_1^{(i)}, K_2^{(i)}, i, \hat{y}])$ (see Fig. 54)
2. $\mathsf{DP}_i \xleftarrow{\$} \mathsf{iO}(\mathbb{1}^\lambda, \mathsf{DProg}_1^0[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j \neq i}, K_1^{(i)}, \hat{y}])$ (see Fig. 55)
3. Output $U_i := (\mathsf{hk}_i, \mathsf{EP}_i, \mathsf{DP}_i, \pi_i, \pi_i')$.

Hybrid $\hat{y}$.1. In this hybrid, we modify the encryption program EP_i sent in the ι-th $\mathsf{NewSession}$ query. In particular, instead of obfuscating EProg^0 (see Fig. 54), we obfuscate EProg^1 (see Fig. 56). In the latter, the PPRF key $K_2^{(i)}$ will now be punctured in position $\hat{y}$. Furthermore, we store into EP_i, the pair

$$(\widehat{\mathsf{pk}}_i, \hat{c}_i) \leftarrow \mathsf{EProg}[K_1^{(i)}, K_2^{(i)}, i](\hat{y}).$$

When $\hat{y}$ is provided as input, EP_i will directly output $(\widehat{\mathsf{pk}}_i, \hat{c}_i)$. The rest remains as before. Since the input-output behaviour of EProg^1 is the same as for EProg^0, no adversary can distinguish between this hybrid and the previous one under the security of iO.

$\mathsf{EProg}^1[K_1^{(i)}, K_2^{(i)}, i, \hat{y}, \widehat{\mathsf{pk}}, \hat{c}]$

Hard-coded. The PPRF keys $K_1^{(i)}$ and $K_2^{(i)}$, the index i, the hybrid index $\hat{y}$, the public key $\widehat{\mathsf{pk}}$ and the ciphertext $\hat{c}$.
Input. A digest $y \in \{0,1\}^{t(\lambda)}$.

```
1. If $y <_{\mathsf{lex}} \hat{y}$: $(\mathsf{pk}_i, c_i) \leftarrow \mathsf{EProg}_{\mathsf{Ls}}[K_2^{(i)}, i](y)$ (see Fig. 30)
2. If $y = \hat{y}$: $(\mathsf{pk}_i, c_i) \leftarrow (\widehat{\mathsf{pk}}, \hat{c})$
3. Otherwise, $(\mathsf{pk}_i, c_i) \leftarrow \mathsf{EProg}[K_1^{(i)}, K_2^{(i)}, i](y)$ (see Fig. 28)
4. Output $(\mathsf{pk}_i, c_i)$
```

Fig. 56. Hybrid $\hat{y}$.1: the unobfuscated encryption program of party P_i

The operations performed by the challenger in order to compute U_i in the ι-th $\mathsf{NewSession}$ query become the following.

1. $K_2^* \leftarrow F_2.\mathsf{Punct}(K_2^{(i)}, \hat{y})$
2. $s_i \leftarrow F_1(K_1^{(i)}, \hat{y})$
3. $(r_i, r_i', r_i'', \eta_i, \eta_i') \leftarrow F_2(K_2^{(i)}, \hat{y})$
4. $(\widehat{\mathsf{pk}}_i, \widehat{\mathsf{sk}}_i) \leftarrow \mathsf{mkFHE.Gen}(1\!\!1^\lambda, i; r_i)$
5. $\hat{c}_i \leftarrow \mathsf{mkFHE.Enc}(\widehat{\mathsf{pk}}_i, s_i; r_i')$
6. $\mathsf{EP}_i \xleftarrow{\$} \mathsf{iO}(1\!\!1^\lambda, \mathsf{EProg}^1[K_1^{(i)}, K_2^*, i, \hat{y}, \widehat{\mathsf{pk}}_i, \hat{c}_i])$ (see Fig. 56)
7. $\mathsf{DP}_i \xleftarrow{\$} \mathsf{iO}(1\!\!1^\lambda, \mathsf{DProg}_1^0[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j \neq i}, K_1^{(i)}, \hat{y}])$ (see Fig. 55)
8. Output $U_i := (\mathsf{hk}_i, \mathsf{EP}_i, \mathsf{DP}_i, \pi_i, \pi_i')$.

Hybrid $\hat{y}$.2. In this hybrid, we modify the decryption program DP_i sent in the ι-th $\mathsf{NewSession}$ query. In particular, instead of obfuscating DProg_1^0 (see Fig. 55), we obfuscate DProg_1^1 (see Fig. 57). In the latter, the PPRF key $K_2^{(i)}$ will now be punctured in position $\hat{y}$. Now, there are two cases: if there exists a tuple $(\mathsf{hk}_j, \mathsf{EP}_j)_{j \neq i} \in \Omega$ such that $\mathsf{Hash}\big(\mathsf{hk}_i, (\mathsf{hk}_j, \mathsf{EP}_j)_{j \neq i}\big) = \hat{y}$, then, we store into DP_i, the partial decryption

$$\hat{d}_i \leftarrow \mathsf{DProg}_1[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j \neq i}]\Big((\mathsf{hk}_j, \mathsf{EP}_j)_{j \neq i}\Big).$$

Notice that, thanks to the subexponential security of Hash, the tuple $(\mathsf{hk}_j, \mathsf{EP}_j)_{j \neq i}$ is univocally defined. If instead, the tuple $(\mathsf{hk}_j, \mathsf{EP}_j)_{j \neq i}$ we are looking for does not exist, we set $\hat{d}_i \leftarrow \perp$. When the hash of the input collides with $\hat{y}$, DP_i will now directly output $\hat{d}_i$. The rest remains as before. Observe that the input-output behaviour of DProg_1^1 is the same as for DProg_1^0. Indeed, the input-output behaviour can change only if the input consists of a tuple $(\mathsf{hk}_j, \mathsf{EP}_j)_{j \neq i}$ that hashes to $\hat{y}$. We know that there exists at most one such tuple in Ω and, in that case, the output of both DProg_1^1 and DProg_1^0 is the hardcoded value $\hat{d}_i$. When $(\mathsf{hk}_j, \mathsf{EP}_j)_{j \neq i}$ in not in Ω, then both DProg_1^1 and DProg_1^0 output $\perp$ as the extraction of the witness from the NIZKs will always fails. We conclude that no adversary can distinguish between this hybrid and the previous one under the security of iO.

The operations performed by the challenger in order to compute U_i in the ι-th $\mathsf{NewSession}$ query become the following. Below, $(\mathsf{hk}_j, \mathsf{EP}_j)_{j \neq i}$ denotes the tuple in Ω that hashes to $\hat{y}$ under hk_i. If such tuple does not exist, we ignore steps 7-10 below, and we simply set $\hat{d}_i \leftarrow \perp$.

1. $K_2^* \leftarrow F_2.\mathsf{Punct}(K_2^{(i)}, \hat{y})$
2. $s_i \leftarrow F_1(K_1^{(i)}, \hat{y})$
3. $(r_i, r_i', r_i'', \eta_i, \eta_i') \leftarrow F_2(K_2^{(i)}, \hat{y})$
4. $(\widehat{\mathsf{pk}}_i, \widehat{\mathsf{sk}}_i) \leftarrow \mathsf{mkFHE.Gen}(1\!\!1^\lambda, i;\, r_i)$
5. $\hat{c}_i \leftarrow \mathsf{mkFHE.Enc}(\widehat{\mathsf{pk}}_i, s_i;\, r_i')$
6. $\mathsf{EP}_i \xleftarrow{\$} \mathsf{iO}(1\!\!1^\lambda, \mathsf{EProg}^1[K_1^{(i)}, K_2^*, i, \hat{y}, \widehat{\mathsf{pk}}_i, \hat{c}_i])$ (see Fig. 56)
7. $\forall j \in [n]: \quad y_j \leftarrow \mathsf{Hash}\big(\mathsf{hk}_j, (\mathsf{hk}_l, \mathsf{EP}_l)_{l \neq j}\big)$
8. $\forall j \in [n]: \quad (\mathsf{pk}_j, c_j) \leftarrow \mathsf{EP}_j(y_j)$
9. $C \leftarrow \mathsf{mkFHE.Eval}\big(\tilde{\mathcal{D}}, \mathsf{pk}_1, c_1, \ldots, \mathsf{pk}_n, c_n\big)$ (see bottom of Fig. 29)
10. $\hat{d}_i \leftarrow \mathsf{mkFHE.PartDec}\Big(C, (\mathsf{pk}_1, \mathsf{pk}_2, \ldots, \mathsf{pk}_n), i, \widehat{\mathsf{sk}}_i;\, \eta_i\Big)$
11. $\mathsf{DP}_i \xleftarrow{\$} \mathsf{iO}(1\!\!1^\lambda, \mathsf{DProg}_1^1[i, \mathsf{sid}, K_2^*, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j \neq i}, K_1^{(i)}, \hat{y}, \hat{d}_i])$ (see Fig. 57)
12. Output $U_i := (\mathsf{hk}_i, \mathsf{EP}_i, \mathsf{DP}_i, \pi_i, \pi_i')$.

Hybrid $\hat{y}$.3. In this hybrid, in the ι-th $\mathsf{NewSession}$ query, we generate $\widehat{\mathsf{pk}}_i$, $\hat{c}_i$ and $\hat{d}_i$ by inputting full-entropy randomness r_i, r_i' and η_i into $\mathsf{mkFHE.Gen}$, $\mathsf{mkFHE.Enc}$ and $\mathsf{mkFHE.PartDec}$ instead of producing it using $F_2(K_2^{(i)}, \hat{y})$. This hybrid is indistinguishable from the previous one by the security of puncturable PRFs.

Remark 2. Observe that since the number of pairs (i, hk_i) is finite, for every $\lambda \in \mathbb{N}$, there exists one that maximises the advantage of the adversary in distinguishing this hybrid from the previous one. We call the corresponding hash key "the worst hash key". Of course the worst hash key is chosen among those for which there exist no collisions in Ω. In the reduction to the security of puncturable PRFs, we can assume that the new adversary (the one attacking F_2) obtains $\hat{y}$, the worst hash key $\widehat{\mathsf{hk}}_i$ and the tuple $(\mathsf{hk}_j, \mathsf{EP}_j)_{j \neq i}$ in Ω that is hashed to $\hat{y}$ (if such tuple exists) as part of its non-uniform advice. The new adversary will simulate the indistinguishability game between Hybrid $\hat{y}$.2 and Hybrid $\hat{y}$.3 using these values.

The operations performed by the challenger in order to compute U_i in the ι-th $\mathsf{NewSession}$ query become the following. Below, $(\mathsf{hk}_j, \mathsf{EP}_j)_{j \neq i}$ denotes the tuple in Ω that hashes to $\hat{y}$ under hk_i. If such tuple does not exist, we ignore steps 6-9 below, and we simply set $\hat{d}_i \leftarrow \perp$.

1. $K_2^* \leftarrow F_2.\mathsf{Punct}(K_2^{(i)}, \hat{y})$
2. $s_i \leftarrow F_1(K_1^{(i)}, \hat{y})$

$\mathsf{DProg}_1^1[i, \mathsf{sid}, K_2^*, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j \neq i}, K_1^{(i)}, \omega, \hat{d}_i]$

Hard-coded. The index i of the party, the session identity sid, a punctured PRF key K_2^*, the encryption program EP_i, the hash key hk_i, the extractable NIZK CRS σ and the extraction trapdoors $(\tau_e^j)_{j \neq i}$, the PPRF key $K_1^{(i)}$, the hybrid index $\hat{y}$, the partial decryption $\hat{d}_i$.
Input. Set of $n-1$ tuples $(\mathsf{hk}_j, \mathsf{EP}_j, \pi_j)_{j \neq i}$.

```
1. If $\mathsf{Hash}\big(\mathsf{hk}_i, (\mathsf{hk}_j, \mathsf{EP}_j)_{j \neq i}\big) <_{\mathsf{lex}} \hat{y}$:
   $d_i \leftarrow \mathsf{DProg}_2[i, \mathsf{sid}, K_2^*, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j \neq i}, K_1^{(i)}]\Big((\mathsf{hk}_j, \mathsf{EP}_j, \pi_j)_{j \neq i}\Big)$ (see Fig. 33)
2. If $\mathsf{Hash}\big(\mathsf{hk}_i, (\mathsf{hk}_j, \mathsf{EP}_j)_{j \neq i}\big) = \hat{y}$ :
   (a) $\forall j \neq i : \quad b_j \leftarrow \mathsf{NIZK.Verify}\big(\sigma, \pi_j, (j, \mathsf{hk}_j, \mathsf{EP}_j)\big)$
   (b) $\forall j \neq i : \quad \big(K_1^{(j)}, K_2^{(j)}\big) \leftarrow \mathsf{NIZK.Extract}\big(\tau_e^j, \pi_j, (j, \mathsf{hk}_j, \mathsf{EP}_j)\big)$ [a]
   (c) If $\exists j \neq i$ such that $b_j = 0$ or $\big(K_1^{(j)}, K_2^{(j)}\big) = \perp$, output $\perp$
   (d) $d_i \leftarrow \hat{d}_i$
3. Otherwise,
   $d_i \leftarrow \mathsf{DProg}_1[i, \mathsf{sid}, K_2^*, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j \neq i}]\Big((\mathsf{hk}_j, \mathsf{EP}_j, \pi_j)_{j \neq i}\Big)$ (see Fig. 32)
4. Output $d_i$
```

Fig. 57. Hybrid $\hat{y}$.2: the unobfuscated decryption program of party P_i

```
3. $(\widehat{\mathsf{pk}}_i, \widehat{\mathsf{sk}}_i) \xleftarrow{\$} \mathsf{mkFHE.Gen}(1\!\!1^\lambda, i)$
4. $\hat{c}_i \xleftarrow{\$} \mathsf{mkFHE.Enc}(\widehat{\mathsf{pk}}_i, s_i)$
5. $\mathsf{EP}_i \xleftarrow{\$} \mathsf{iO}(1\!\!1^\lambda, \mathsf{EProg}^1[K_1^{(i)}, K_2^*, i, \hat{y}, \widehat{\mathsf{pk}}_i, \hat{c}_i])$ (see Fig. 56)
6. $\forall j \in [n] : \quad y_j \leftarrow \mathsf{Hash}\big(\mathsf{hk}_j, (\mathsf{hk}_l, \mathsf{EP}_l)_{l \neq j}\big)$
7. $\forall j \in [n] : \quad (\mathsf{pk}_j, c_j) \leftarrow \mathsf{EP}_j(y_j)$
8. $C \leftarrow \mathsf{mkFHE.Eval}\big(\tilde{\mathcal{D}}, \mathsf{pk}_1, c_1, \ldots, \mathsf{pk}_n, c_n\big)$ (see bottom of Fig. 29)
9. $\hat{d}_i \xleftarrow{\$} \mathsf{mkFHE.PartDec}\Big(C, (\mathsf{pk}_1, \mathsf{pk}_2, \ldots, \mathsf{pk}_n), i, \widehat{\mathsf{sk}}_i\Big)$
10. $\mathsf{DP}_i \xleftarrow{\$} \mathsf{iO}(1\!\!1^\lambda, \mathsf{DProg}_1^1[i, \mathsf{sid}, K_2^*, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j \neq i}, K_1^{(i)}, \hat{y}, \hat{d}_i])$ (see Fig. 57)
11. Output $U_i := (\mathsf{hk}_i, \mathsf{EP}_i, \mathsf{DP}_i, \pi_i, \pi_i')$.
```

Hybrid $\hat{y}$.4. In this hybrid, in the ι-th $\mathsf{NewSession}$ query, instead of computing $\widehat{\mathsf{pk}}_i$, $\hat{c}_i$ and $\hat{d}_i$ using $\mathsf{mkFHE.Gen}$, $\mathsf{mkFHE.Enc}$ and $\mathsf{mkFHE.PartDec}$, we simulate them. Notice that the multi-key FHE simulator $\mathsf{mkFHE.Sim}_2$ needs to receive the inputs and the randomness used by the other parties. We retrieve the latter by expanding the PRF keys $K_1^{(j)}$ and $K_2^{(j)}$ hidden in EP_j (we recall that $(\mathsf{hk}_j, \mathsf{EP}_j)_{j \neq i}$ denotes the only tuple in Ω that is hashed to $\hat{y}$ under hk_i). Since the obfuscation scheme is injective, $K_1^{(j)}$ and $K_2^{(j)}$ are univocally defined.

This hybrid is indistinguishable from the previous one under the reusable semi-malicious security of multi-key FHE. For the reduction, we use the same trick as in Hybrid $\hat{y}$.3, i.e. we provide the adversary with $\hat{y}$, the worst hash key, the only preimage $(\mathsf{hk}_j, \mathsf{EP}_j)_{j\neq i}$ of $\hat{y}$ in Ω along with the PRF keys $K_1^{(j)}, K_2^{(j)}$ hidden in each EP_j as part of the non-uniform advice string.

The operations performed by the challenger in order to compute U_i in the ι-th $\mathsf{NewSession}$ query become the following. Below, $(\mathsf{hk}_j, \mathsf{EP}_j)_{j\neq i}$ denotes the tuple in Ω that hashes to $\hat{y}$ under hk_i. For every $j \neq i$, we use $K_1^{(j)}$ and $K_2^{(j)}$ to denote the PPRF keys hidden in EP_j. If such tuple does not exist, we ignore steps 4-8 below, and we simply set $\hat{d}_i \leftarrow \perp$.

```
 1. $K_2^* \leftarrow F_2.\mathsf{Punct}(K_2^{(i)}, \hat{y})$
 2. $(\phi, \widehat{\mathsf{pk}}_i, \hat{c}_i) \xleftarrow{\$} \mathsf{mkFHE.Sim}_1(1\!\!1^\lambda, i)$
 3. $\mathsf{EP}_i \xleftarrow{\$} \mathsf{iO}(1\!\!1^\lambda, \mathsf{EProg}^1[K_1^{(i)}, K_2^*, i, \hat{y}, \widehat{\mathsf{pk}}_i, \hat{c}_i])$ (see Fig. 56)
 4. $\forall j \in [n]: \quad y_j \leftarrow \mathsf{Hash}\big(\mathsf{hk}_j, (\mathsf{hk}_l, \mathsf{EP}_l)_{l\neq j}\big)$
 5. $\forall j \in [n]: \quad s_j \leftarrow F_1(K_1^{(j)}, y_j)$
 6. $\forall j \neq i: \quad (r_j, r'_j, r''_j, \eta_j, \eta'_j) \leftarrow F_2(K_2^{(j)}, y_j)$
 7. $\hat{R} \leftarrow \mathcal{D}(1\!\!1^\lambda;\ s_1 \oplus \cdots \oplus s_n)$
 8. $\hat{d}_i \xleftarrow{\$} \mathsf{mkFHE.Sim}_2\Big(\phi, \widetilde{dist}, \hat{R}, (s_j, r_j, r'_j)_{j\neq i}\Big)$
 9. $\mathsf{DP}_i \xleftarrow{\$} \mathsf{iO}(1\!\!1^\lambda, \mathsf{DProg}_1^1[i, \mathsf{sid}, K_2^*, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j\neq i}, K_1^{(i)}, \hat{y}, \hat{d}_i])$ (see Fig. 57)
10. Output $U_i := (\mathsf{hk}_i, \mathsf{EP}_i, \mathsf{DP}_i, \pi_i, \pi'_i)$.
```

Hybrid $\hat{y}$.5. In this hybrid, in the ι-th $\mathsf{NewSession}$ query, we generate $\widehat{\mathsf{pk}}_i$, $\hat{c}_i$ and $\widehat{d}_i$ using the randomness generated by $F_2(K_2^{(i)}, \hat{y})$. This hybrid is indistinguishable from the previous one under the security of the puncturable PRF.

The operations performed by the challenger in order to compute U_i in the ι-th $\mathsf{NewSession}$ query become the following. Below, $(\mathsf{hk}_j, \mathsf{EP}_j)_{j\neq i}$ denotes the tuple in Ω that hashes to $\hat{y}$ under hk_i. For every $j \neq i$, we use $K_1^{(j)}$ and $K_2^{(j)}$ to denote the PPRF keys hidden in EP_j. If such tuple does not exist, we ignore steps 5-9 below, and we simply set $\hat{d}_i \leftarrow \perp$.

```
 1. $K_2^* \leftarrow F_2.\mathsf{Punct}(K_2^{(i)}, \hat{y})$
 2. $(r_i, r'_i, r''_i, \eta_i, \eta'_i) \leftarrow F_2(K_2^{(i)}, \hat{y})$
 3. $(\phi, \widehat{\mathsf{pk}}_i, \hat{c}_i) \leftarrow \mathsf{mkFHE.Sim}_1(1\!\!1^\lambda, i;\ r''_i)$
 4. $\mathsf{EP}_i \xleftarrow{\$} \mathsf{iO}(1\!\!1^\lambda, \mathsf{EProg}^1[K_1^{(i)}, K_2^*, i, \hat{y}, \widehat{\mathsf{pk}}_i, \hat{c}_i])$ (see Fig. 56)
 5. $\forall j \in [n]: \quad y_j \leftarrow \mathsf{Hash}\big(\mathsf{hk}_j, (\mathsf{hk}_l, \mathsf{EP}_l)_{l\neq j}\big)$
 6. $\forall j \in [n]: \quad s_j \leftarrow F_1(K_1^{(j)}, y_j)$
 7. $\forall j \neq i: \quad (r_j, r'_j, r''_j, \eta_j, \eta'_j) \leftarrow F_2(K_2^{(j)}, y_j)$
 8. $\hat{R} \leftarrow \mathcal{D}(1\!\!1^\lambda;\ s_1 \oplus \cdots \oplus s_n)$
 9. $\hat{d}_i \leftarrow \mathsf{mkFHE.Sim}_2\Big(\phi, \widetilde{dist}, \hat{R}, (s_j, r_j, r'_j)_{j\neq i};\ \eta'_i\Big)$
10. $\mathsf{DP}_i \xleftarrow{\$} \mathsf{iO}(1\!\!1^\lambda, \mathsf{DProg}_1^1[i, \mathsf{sid}, K_2^*, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j\neq i}, K_1^{(i)}, \hat{y}, \hat{d}_i])$ (see Fig. 57)
11. Output $U_i := (\mathsf{hk}_i, \mathsf{EP}_i, \mathsf{DP}_i, \pi_i, \pi'_i)$.
```

Hybrid $\hat{y}$.6. In this hybrid, we change the encryption program EP_i sent in the ι-th $\mathsf{NewSession}$ query. In particular, we switch back to an obfuscation of

EProg^0 (see Fig. 54). This time, however, instead of hardcoding $\hat{y}$, we hardcode the next element in $\{0,1\}^{t(\lambda)}$. We denote it by $\hat{y}'$[18]. The input-output behaviour of EP_i remains the same as in the previous hybrid, so we can argue for indistinguishability based on the security of iO.

The operations performed by the challenger in order to compute U_i in the ι-th $\mathsf{NewSession}$ query become the following. Below, $(\mathsf{hk}_j, \mathsf{EP}_j)_{j \neq i}$ denotes the tuple in Ω that hashes to $\hat{y}$ under hk_i. For every $j \neq i$, we use $K_1^{(j)}$ and $K_2^{(j)}$ to denote the PPRF keys hidden in EP_j. If such tuple does not exist, we ignore steps 5-9 below, and we simply set $\hat{d}_i \leftarrow \bot$.

1. $K_2^* \leftarrow F_2.\mathsf{Punct}(K_2^{(i)}, \hat{y})$
2. $(r_i, r'_i, r''_i, \eta_i, \eta'_i) \leftarrow F_2(K_2^{(i)}, \hat{y})$
3. $(\phi, \widehat{\mathsf{pk}}_i, \hat{c}_i) \leftarrow \mathsf{mkFHE.Sim}_1(\mathbb{1}^\lambda, i; r''_i)$
4. $\mathsf{EP}_i \xleftarrow{\$} \mathsf{iO}(\mathbb{1}^\lambda, \mathsf{EProg}^0[K_1^{(i)}, K_2^{(i)}, i, \hat{y}'])$ (see Fig. 54)
5. $\forall j \in [n]: \quad y_j \leftarrow \mathsf{Hash}\big(\mathsf{hk}_j, (\mathsf{hk}_l, \mathsf{EP}_l)_{l \neq j}\big)$
6. $\forall j \in [n]: \quad s_j \leftarrow F_1(K_1^{(j)}, y_j)$
7. $\forall j \neq i: \quad (r_j, r'_j, r''_j, \eta_j, \eta'_j) \leftarrow F_2(K_2^{(j)}, y_j)$
8. $\hat{R} \leftarrow \mathcal{D}(\mathbb{1}^\lambda; s_1 \oplus \cdots \oplus s_n)$
9. $\hat{d}_i \leftarrow \mathsf{mkFHE.Sim}_2\Big(\phi, \widetilde{dist}, \hat{R}, (s_j, r_j, r'_j)_{j \neq i}; \eta'_i\Big)$
10. $\mathsf{DP}_i \xleftarrow{\$} \mathsf{iO}(\mathbb{1}^\lambda, \mathsf{DProg}_1^1[i, \mathsf{sid}, K_2^*, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j \neq i}, K_1^{(i)}, \hat{y}, \hat{d}_i])$ (see Fig. 57)
11. Output $U_i := (\mathsf{hk}_i, \mathsf{EP}_i, \mathsf{DP}_i, \pi_i, \pi'_i)$.

Hybrid $\hat{y}$.7. In this hybrid, we change the decryption program DP_i sent in the ι-th $\mathsf{NewSession}$ query. In particular, we switch back to an obfuscation of DProg_1^0 (see Fig. 55). This time, however, instead of hardcoding $\hat{y}$, we hardcode $\hat{y}'$[19]. The input-output behaviour of DP_i remains the same as in the previous hybrid, so we can argue for indistinguishability based on the security of iO.

We observe that the differing-inputs can only consist of tuples $(\mathsf{hk}_j, \mathsf{EP}_j)_{j \neq i}$ that hash to $\hat{y}$. Any of these tuples that does not belong to Ω is mapped to $\bot$ by DP_i in both hybrids. Indeed, the extraction of the witnesses from the proofs will always fail. The only differing input can therefore be the only preimage of $\hat{y}$ in Ω, if this exists. However, even for such input, DP_i behaves the same in the two hybrids.

The operations performed by the challenger in order to compute U_i in the ι-th $\mathsf{NewSession}$ query become the following.

1. $\mathsf{EP}_i \xleftarrow{\$} \mathsf{iO}(\mathbb{1}^\lambda, \mathsf{EProg}^0[K_1^{(i)}, K_2^{(i)}, i, \hat{y}'])$ (see Fig. 54)
2. $\mathsf{DP}_i \xleftarrow{\$} \mathsf{iO}(\mathbb{1}^\lambda, \mathsf{DProg}_1^0[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j \neq i}, K_1^{(i)}, \hat{y}'])$ (see Fig. 55)
3. Output $U_i := (\mathsf{hk}_i, \mathsf{EP}_i, \mathsf{DP}_i, \pi_i, \pi'_i)$.

[18] If $\hat{y}$ is already the maximum of $\{0,1\}^{t(\lambda)}$, we augment $\{0,1\}^{t(\lambda)}$ with an imaginary element that is strictly greater than all other values. Let $\hat{y}'$ be such value.

[19] If $\hat{y}$ is already the maximum of $\{0,1\}^{t(\lambda)}$, we augment $\{0,1\}^{t(\lambda)}$ with an imaginary element that is strictly greater than all other values. Let $\hat{y}'$ be such value.

When $\hat{y}$ reaches the maximum in $\{0,1\}^{t(\lambda)}$, Hybrid $\hat{y}.7$ is indistinguishable from Hybrid $4.\iota.3$ due to the security of iO. Indeed, in Hybrid $\hat{y}.7$, for any input, EP_i computes the output using $\mathsf{EProg}_{\mathsf{Ls}}$, whereas DP_i computes the output using DProg_2. This terminates the proof of the claim. ■

Hybrid 4.ι.4. In this hybrid, we change the decryption program DP_i generated in the ι-th $\mathsf{NewSession}$ query, switching to an obfuscation of $\mathsf{DProg}_{\mathsf{Ls}}$ (see Fig. 31). In the latter, we embed the ELF f used to reply to the first $\iota - 1$ $\mathsf{NewSession}$ queries, and a random PPRF key K.

The operations performed by the challenger in order to compute U_i in the ι-th $\mathsf{NewSession}$ query become the following.

1. $K \xleftarrow{\$} F.\mathsf{Gen}(\mathbb{1}^\lambda)$
2. $\mathsf{EP}_i \xleftarrow{\$} \mathsf{iO}(\mathbb{1}^\lambda, \mathsf{EProg}_{\mathsf{Ls}}[K_2^{(i)}, i])$ (see Fig. 30)
3. $\mathsf{DP}_i \xleftarrow{\$} \mathsf{iO}(\mathbb{1}^\lambda, \mathsf{DProg}_{\mathsf{Ls}}[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j \neq i}, K, f])$ (see Fig. 31)
4. Output $U_i := (\mathsf{hk}_i, \mathsf{EP}_i, \mathsf{DP}_i, \pi_i, \pi'_i)$.

In the hybrid, we also change the reply to the sampling queries concerning the ι-th session.In particular, when the NIZKs verify and we succeed in extracting the witnesses from the messages $(U_j)_{j \neq i}$ provided by the adversary, we answer the sampling query as follows.

1. $z \leftarrow f\big((\mathsf{hk}_j, \mathsf{EP}_j)_{j \in [n]}\big)$
2. $s \leftarrow F(K, z)$
3. $R \leftarrow \mathcal{D}(\mathbb{1}^\lambda; s)$
4. Provide R to the adversary.

Claim. Assuming the subexponential security of iO, of the puncturable PRFs F and F_1 and the subexponential collision intractability of the hash function, no PPT adversary can distinguish between Hybrid $4.\iota.3$ and Hybrid $4.\iota.4$.

Proof of the claim. We select the security parameter of the subexponentially collision resistance hash function so that, for any PPT adversary,

$$2^{2\lambda \cdot (n-1)} \cdot \mathsf{Adv}^{\mathcal{A}}_{\mathsf{CR}}(\lambda) = \mathsf{negl}(\lambda).$$

Observe that $|\Omega| = 2^{\lambda \cdot (n-1)}$. We conclude that with overwhelming probability over hk_i, there exist no collisions in Ω. Otherwise, the adversary that simply outputs two random elements in Ω would break the above assumption. We can therefore, prove indistinguishability conditioned on this event occurring.

We proceed once again through a series of indistinguishable hybrids. Their number will be superpolynomial. In particular, we repeat the following sequence for every $\omega \in \Omega$ (Ω was defined in the proof of Claim B). We initially set ω to be the minimum in Ω according to the lexicographical order. Then, we gradually increment it until we reach the maximum. In the proof, we use $\overline{\omega}$ to denote the tuple $(\mathsf{hk}_j, \mathsf{EP}_j)_{j \in [n]}$ where $(\mathsf{hk}_i, \mathsf{EP}_i)$ are the hash key and the encryption program chosen by party P_i, and $(\mathsf{hk}_j, \mathsf{EP}_j)_{j \neq i} = \omega$.

$\mathsf{DProg}_2^0[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j\neq i}, K_1^{(i)}, K, f, \omega]$

Hard-coded. The index i of the party, the session identity sid, a PPRF key K_2, the encryption program EP_i, the hash key hk_i, the extractable NIZK CRS σ and the extraction trapdoors $(\tau_e^j)_{j\neq i}$, the PPRF key $K_1^{(i)}$, the PPRF key K, the ELF f, the hybrid index ω.
Input. Set of $n-1$ tuples $(\mathsf{hk}_j, \mathsf{EP}_j, \pi_j)_{j\neq i}$.

1. If $(\mathsf{hk}_j, \mathsf{EP}_j)_{j\neq i} <_{\mathsf{lex}} \omega$:
   $d_i \leftarrow \mathsf{DProg}_{\mathsf{Ls}}[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j\neq i}, K, f]\Big((\mathsf{hk}_j, \mathsf{EP}_j, \pi_j)_{j\neq i}\Big)$ (see Fig. 31)
2. Otherwise,
   $d_i \leftarrow \mathsf{DProg}_2[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j\neq i}, K_1^{(i)}]\Big((\mathsf{hk}_j, \mathsf{EP}_j, \pi_j)_{j\neq i}\Big)$ (see Fig. 33)
3. Output d_i

Fig. 58. Hybrid $\omega.0$: the unobfuscated decryption program of party P_i

Hybrid ω.0. In this hybrid, we modify the decryption program DP_i sent in the ι-th $\mathsf{NewSession}$ query, switching to an obfuscation of DProg_2^0 (see Fig. 58). The new program will have the hybrid index ω hardcoded. Whenever the input $(\mathsf{hk}_j, \mathsf{EP}_j)_{j\neq i}$ is strictly smaller than ω according to the lexicographical order, DP_i will compute the output using $\mathsf{DProg}_{\mathsf{Ls}}$ (see Fig. 31), otherwise it will use DProg_2 (see Fig. 33). Also the answer to the sampling queries is modified: if the adversary queries messages $(U_j)_{j\neq i}$ for the ι-th session such that $(\mathsf{hk}_j, \mathsf{EP}_j)_{j\neq i}$ is strictly smaller than ω, the challenger replies as in Hybrid 4.ι.4. In the other cases, it replies as in Hybrid 4.ι.3.

When ω is the minimum in $\varOmega$, Hybrid $\omega.0$ is indistinguishable from Hybrid 4.ι.3 by the security of obfuscation. Indeed, all $(\mathsf{hk}_j, \mathsf{EP}_j)_{j\neq i}$ that are strictly smaller than ω contain a malformed pair $(\mathsf{hk}_j, \mathsf{EP}_j)$. Since the NIZK extraction fails, DProg_2 always outputs $\perp$ in these cases. The same does DP_i in Hybrid 4.ι.4. Clearly, the programs behave identically when $(\mathsf{hk}_j, \mathsf{EP}_j)_{j\neq i} \geq_{\mathsf{lex}} \omega$. When ω in not the minimum in $\varOmega$, instead, this hybrid is identical to the previous one, i.e. Hybrid $\widehat{\omega}.4$ where $\widehat{\omega}$ is the previous value of ω.

The operations performed by the challenger in order to compute U_i in the ι-th $\mathsf{NewSession}$ query become the following.

1. $K \xleftarrow{\$} F.\mathsf{Gen}(1\!\!1^\lambda)$
2. $\mathsf{EP}_i \xleftarrow{\$} \mathsf{iO}(1\!\!1^\lambda, \mathsf{EProg}_{\mathsf{Ls}}[K_2^{(i)}, i])$ (see Fig. 30)
3. $\mathsf{DP}_i \xleftarrow{\$} \mathsf{iO}(1\!\!1^\lambda, \mathsf{DProg}_2^0[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j\neq i}, K_1^{(i)}, K, f, \omega])$ (see Fig. 58)
4. Output $U_i := (\mathsf{hk}_i, \mathsf{EP}_i, \mathsf{DP}_i, \pi_i, \pi_i')$.

Hybrid ω.1. In this hybrid, we modify the decryption program DP_i sent in the ι-th $\mathsf{NewSession}$ query, switching to an obfuscation of DProg_2^1 (see Fig. 59). The keys $K_1^{(i)}$ and K stored in DP_i will now be punctured in $y_i = \mathsf{Hash}(\mathsf{hk}_i, \omega)$

and $f(\overline{\varpi})$, respectively. We also hardcode the value $\hat{R}$ that DProg_2 feeds into the partial decryption simulator when ω is given as input. The behaviour of DP_i remains as in the previous hybrid, with the exception that when $(\mathsf{hk}_j, \mathsf{EP}_j)_{j \neq i} = \omega$ and the NIZK extraction succeeds, the program directly feeds the hardcoded $\hat{R}$ into the partial decryption simulator. We highlight that, the modified program will never need to evaluate $K_1^{(i)}$ in y_i. Indeed, by the subexponential collision resistance of the hash function, the only preimage of y_i in $\varOmega$ will be ω. Moreover, K will never be evaluated in $f(\overline{\varpi})$ as the ELF is set in injective mode. Since the input-output behaviour of DP_i has not changed (notice that all the witnesses for the NIZK statement lead to the same sk_i by the injectivity of iO), this hybrid and the previous one are indistinguishable under the security of iO.

The operations performed by the challenger in order to compute U_i in the ι-th $\mathsf{NewSession}$ query become the following. Below, we rewrite ω as $(\mathsf{hk}_j, \mathsf{EP}_j)_{j \neq i}$. We denote the first PRF key hardcoded in EP_j by $K_1^{(j)}$ (we recall that we are only considering EP_js that are well-formed). For the reduction, since ω is fixed and the adversary is non-uniform, we can assume it knows $K_1^{(j)}$ for every $j \neq i$ (the latter is uniquely determined by ω by the injectivity of iO).

1. $K \xleftarrow{\$} F.\mathsf{Gen}(\mathbb{1}^\lambda)$
2. $\mathsf{EP}_i \xleftarrow{\$} \mathsf{iO}(\mathbb{1}^\lambda, \mathsf{EProg}_{\mathsf{Ls}}[K_2^{(i)}, i])$ (see Fig. 30)
3. $z \leftarrow f(\overline{\varpi})$
4. $K^* \leftarrow F.\mathsf{Punct}(K, z)$
5. $\forall j \in [n]: \quad y_j \leftarrow \mathsf{Hash}\big(\mathsf{hk}_j, (\mathsf{hk}_l, \mathsf{EP}_l)_{l \neq j}\big)$
6. $K_1^* \leftarrow F_1.\mathsf{Punct}(K_1^{(i)}, y_i)$
7. $\forall j \in [n]: \quad s_j \leftarrow F_1(K_1^{(j)}, y_j)$
8. $\hat{R} \leftarrow \mathcal{D}(\mathbb{1}^\lambda; s_1 \oplus s_2 \oplus \cdots \oplus s_n)$
9. $\mathsf{DP}_i \xleftarrow{\$} \mathsf{iO}(\mathbb{1}^\lambda, \mathsf{DProg}_2^1[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j \neq i}, K_1^*, K^*, f, \omega, \hat{R}])$ (see Fig. 59)
10. Output $U_i := (\mathsf{hk}_i, \mathsf{EP}_i, \mathsf{DP}_i, \pi_i, \pi_i')$.

Hybrid ω.2. In this hybrid, in the ι-th $\mathsf{NewSession}$ query, we generate $\hat{R}$ using true randomness instead of using $s_1 \oplus \cdots \oplus s_n$ where $s_j \leftarrow F_1(K_1^{(j)}, y_j)$. Furthermore, if the adversary issues any sampling queries $(U_j)_{j \neq i}$ for the ι-th session where the NIZKs verify, the extraction succeeds and $(\mathsf{hk}_j, \mathsf{EP}_j)_{j \neq i}$ coincides with ω, the challenger replies with $\hat{R}$.

Indistinguishability between this hybrid and the previous one is a consequence of the security of the puncturable PRF F_1. Indeed, we are able to substitute s_i with a truly random string without the adversary noticing it. Furthermore, observe that the challenger never needs to evaluate F_1 over $y_i := \mathsf{Hash}(\mathsf{hk}_i, \omega)$. Indeed, with overwhelming probability, there exists no pair of well-formed tuples $(\mathsf{hk}_j, \mathsf{EP}_j)_{j \neq i}$ having y_i as digest.

The operations performed by the challenger in order to compute U_i in the ι-th $\mathsf{NewSession}$ query become the following. Below, we rewrite ω as $(\mathsf{hk}_j, \mathsf{EP}_j)_{j \neq i}$. We denote the first PRF key hardcoded in EP_j by $K_1^{(j)}$ (we recall that we are only considering EP_js that are well-formed). For the reduction, since ω is fixed

$\mathsf{DProg}_2^1[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j \neq i}, K_1^*, K^*, f, \omega, \hat{R}]$

Hard-coded. The index i of the party, the session identity sid, a PPRF key $K_2^{(i)}$, the encryption program EP_i, the hash key hk_i, the extractable NIZK CRS σ and the extraction trapdoors $(\tau_e^j)_{j \neq i}$, the punctured PRF keys K_1^* and K^*, the ELF f, the hybrid index ω, the sample $\hat{R}$.
Input. Set of $n-1$ tuples $(\mathsf{hk}_j, \mathsf{EP}_j, \pi_j)_{j \neq i}$.

```
1. If $(\mathsf{hk}_j, \mathsf{EP}_j)_{j \neq i} <_{\mathsf{lex}} \omega$:
   $d_i \leftarrow \mathsf{DProg}_{\mathsf{Ls}}[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j \neq i}, K^*, f]\Big((\mathsf{hk}_j, \mathsf{EP}_j, \pi_j)_{j \neq i}\Big)$ (see Fig. 31)
2. If $(\mathsf{hk}_j, \mathsf{EP}_j)_{j \neq i} = \omega$ :
   (a) $\forall j \neq i : \quad b_j \leftarrow \mathsf{NIZK.Verify}\big(\sigma, \pi_j, (j, \mathsf{hk}_j, \mathsf{EP}_j)\big)$
   (b) $\forall j \neq i : \quad \big(K_1^{(j)}, K_2^{(j)}\big) \leftarrow \mathsf{NIZK.Extract}\big(\tau_e^j, \pi_j, (j, \mathsf{hk}_j, \mathsf{EP}_j)\big)$ [a]
   (c) If $\exists j \neq i$ such that $b_j = 0$ or $\big(K_1^{(j)}, K_2^{(j)}\big) = \perp$, output $\perp$
   (d) $\forall j \in [n] : \quad y_j \leftarrow \mathsf{Hash}\big(\mathsf{hk}_j, (\mathsf{hk}_l, \mathsf{EP}_l)_{l \neq j}\big)$
   (e) $\forall j \neq i : \quad s_j \leftarrow F_1(K_1^{(j)}, y_j)$
   (f) $\forall j \in [n] : \quad (r_j, r'_j, r''_j, \eta_j, \eta'_j) \leftarrow F_2(K_2^{(j)}, y_j)$
   (g) $(\phi, \mathsf{pk}_i, c_i) \leftarrow \mathsf{mkFHE.Sim}_1(\mathbb{1}^\lambda, i; r''_i)$
   (h) $d_i \leftarrow \mathsf{mkFHE.Sim}_2\Big(\phi, \tilde{\mathcal{D}}, \hat{R}, (s_j, r_j, r'_j)_{j \neq i}; \eta'_i\Big)$ (see bottom of Fig. 29)
3. Otherwise,
   $d_i \leftarrow \mathsf{DProg}_2[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j \neq i}, K_1^*]\Big((\mathsf{hk}_j, \mathsf{EP}_j, \pi_j)_{j \neq i}\Big)$ (see Fig. 33)
4. Output $d_i$
```

Fig. 59. Hybrid $\omega.1$: the unobfuscated decryption program of party P_i

and the adversary is non-uniform, we can assume it knows $K_1^{(j)}$ for every $j \neq i$ (the latter is uniquely determined by ω by the injectivity of iO).

```
1. $K \xleftarrow{\$} F.\mathsf{Gen}(\mathbb{1}^\lambda)$
2. $\mathsf{EP}_i \xleftarrow{\$} \mathsf{iO}(\mathbb{1}^\lambda, \mathsf{EProg}_{\mathsf{Ls}}[K_2^{(i)}, i])$ (see Fig. 30)
3. $z \leftarrow f(\overline{\omega})$
4. $K^* \leftarrow F.\mathsf{Punct}(K, z)$
5. $y_i \leftarrow \mathsf{Hash}(\mathsf{hk}_i, \omega)$
6. $K_1^* \leftarrow F_1.\mathsf{Punct}(K_1^{(i)}, y_i)$
7. $\hat{R} \xleftarrow{\$} \mathcal{D}(\mathbb{1}^\lambda)$
8. $\mathsf{DP}_i \xleftarrow{\$} \mathsf{iO}(\mathbb{1}^\lambda, \mathsf{DProg}_2^1[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j \neq i}, K_1^*, K^*, f, \omega, \hat{R}])$ (see Fig. 59)
9. Output $U_i := (\mathsf{hk}_i, \mathsf{EP}_i, \mathsf{DP}_i, \pi_i, \pi'_i)$.
```

Hybrid ω.3. In this hybrid, in the ι-th $\mathsf{NewSession}$ query, we generate the randomness of $\hat{R}$ using $F(K, z)$ where $z = f(\omega)$. Indistinguishability is a consequence of the security of the puncturable PRF F.

The operations performed by the challenger in order to compute U_i in the ι-th $\mathsf{NewSession}$ query become the following. Below, we rewrite ω as $(\mathsf{hk}_j, \mathsf{EP}_j)_{j \neq i}$.

1. $K \xleftarrow{\$} F.\mathsf{Gen}(\mathbb{1}^\lambda)$
2. $\mathsf{EP}_i \xleftarrow{\$} \mathsf{iO}(\mathbb{1}^\lambda, \mathsf{EProg}_{\mathsf{Ls}}[K_2^{(i)}, i])$ (see Fig. 30)
3. $z \leftarrow f(\overline{\omega})$
4. $K^* \leftarrow F.\mathsf{Punct}(K, z)$
5. $y_i \leftarrow \mathsf{Hash}(\mathsf{hk}_i, \omega)$
6. $K_1^* \leftarrow F_1.\mathsf{Punct}(K_1^{(i)}, y_i)$
7. $s \leftarrow F(K, z)$
8. $\hat{R} \leftarrow \mathcal{D}(\mathbb{1}^\lambda;\, s)$
9. $\mathsf{DP}_i \xleftarrow{\$} \mathsf{iO}(\mathbb{1}^\lambda, \mathsf{DProg}_2^1[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j \neq i}, K_1^*, K^*, f, \omega, \hat{R}])$ (see Fig. 59)
10. Output $U_i := (\mathsf{hk}_i, \mathsf{EP}_i, \mathsf{DP}_i, \pi_i, \pi_i')$.

When the adversary issues any sampling queries $(U_j)_{j \neq i}$ for the ι-th session where the NIZKs verify, the extraction succeeds and $(\mathsf{hk}_j, \mathsf{EP}_j)_{j \neq i}$ coincides with ω, the challenger replies as follows.

1. $z \leftarrow f\big((\mathsf{hk}_j, \mathsf{EP}_j)_{j \in [n]}\big)$
2. $s \leftarrow F(K, z)$
3. $R \leftarrow \mathcal{D}(\mathbb{1}^\lambda;\, s)$

Hybrid ω.4. In this hybrid, we modify the decryption program DP_i sent in the ι-th $\mathsf{NewSession}$ query, switching back to an obfuscation of DProg_2^0 (see Fig. 58). This time, however, the we do not hardcode ω into it, but the next element in Ω. We denote it by ω'[20]. The input-output behaviour of DP_i has not changed since the last hybrid, so, we can argue for indistinguishability under the security of iO.

We observe, indeed, that the behaviour of the program can change only if the input satisfies $\omega \leq_{\mathsf{lex}} (\mathsf{hk}_j, \mathsf{EP}_j)_{j \neq i} <_{\mathsf{lex}} \omega'$. If $\omega <_{\mathsf{lex}} (\mathsf{hk}_j, \mathsf{EP}_j)_{j \neq i} <_{\mathsf{lex}} \omega'$, the programs always output $\bot$ because one $(\mathsf{hk}_j, \mathsf{EP}_j)$ must be malformed, so the NIZK extraction always fails.

We therefore focus on the case $(\mathsf{hk}_j, \mathsf{EP}_j)_{j \neq i} = \omega$. We observe that in this case, the behaviour of the new DP_i is the same as in the previous hybrid. In particular, if the NIZK is not rejected, the value $\hat{R}$ computed by the new DP_i was the same that was previously hardcoded.

The operations performed by the challenger in order to compute U_i in the ι-th $\mathsf{NewSession}$ query become the following. Below, we rewrite ω as $(\mathsf{hk}_j, \mathsf{EP}_j)_{j \neq i}$.

1. $K \xleftarrow{\$} F.\mathsf{Gen}(\mathbb{1}^\lambda)$
2. $\mathsf{EP}_i \xleftarrow{\$} \mathsf{iO}(\mathbb{1}^\lambda, \mathsf{EProg}_1[K_2^{(i)}, i])$ (see Fig. 56)
3. $\mathsf{DP}_i \xleftarrow{\$} \mathsf{iO}(\mathbb{1}^\lambda, \mathsf{DProg}_2^0[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j \neq i}, K_1^{(i)}, K, f, \omega'])$ (see Fig. 58)
4. Output $U_i := (\mathsf{hk}_i, \mathsf{EP}_i, \mathsf{DP}_i, \pi_i, \pi_i')$.

We conclude the proof of the claim by observing that when ω reaches the maximum in Ω, Hybrid ω.4 is indistinguishable from Hybrid 4.ι.4 under the security of iO. Indeed, in Hybrid ω.4, DP_i computed all the outputs running

[20] If ω is already the maximum of Ω, we augment Ω with an imaginary element that is strictly greater than all other values. Let ω' be such value.

$\mathsf{DProg}_{\mathsf{Ls}}$. ■

Hybrid 5. In this hybrid, we modify the sampling queries. In particular, we do not try anymore to extract the witnesses from the NIZKs provided by the adversary, we simply verify the proofs. If the check succeeds, we proceed by inputting $(\mathsf{hk}_j, \mathsf{EP}_j)_{j\in[n]}$ in the ELF, we feed the result into F and we use the output as randomness for $\mathcal{D}(\mathbb{1}^\lambda)$. If the verification fails, we reply with $\bot$.

This hybrid is indistinguishable from hybrid $4.(M+1).4$. Indeed, an adversary can distinguish if and only if, in Hybrid $4.(M+1).4$, it can generate a proof that verifies but cannot be extracted. Such adversary would also be able to distinguish between Hybrid 0 and Hybrid $4.(M+1).4$. However, we proved that such adversary cannot exist.

Hybrid 6. In this hybrid, we switch the ELF f to lossy mode. Let $p'(\lambda)$ be a polynomial upper bound on the running time of the lossy distributed sampler challenger in Hybrid 5 when it interacts with an adversary running in time a most $p(\lambda)$. We choose the polynomial $q(\lambda)$ parametrising the lossy mode so that no adversary running in time at most $p(\lambda) + p'(\lambda)$ can distinguish between the injective mode and the lossy mode with advantage greater than $\delta/2$.

We highlight that Hybrid 5 and Hybrid 6 can be distinguished with non-negligible advantage. However, by the security of ELFs, no adversary running in time at most $p(\lambda)$ can distinguish between them with advantage greater than $\delta/2$.

In order to conclude the proof, we show that it is possible to choose the security parameters of the subexponentially secure primitives so that Claim B and Claim B are all true. This is an immediate consequence of the fact that the dependency graph among subexponentially secure primitives in Fig. 35 contains no cycles.

Regularity. Assume that the ELF is regular. We observe that the output of $\mathsf{Project}$ is either $\bot$ or an element in the image of f. The output of $\mathcal{Z}(\zeta)$ is $\bot$ with probability $1/2$. Otherwise, the output is $f(x)$ where x is uniformly sampled over the domain of f. By the regularity of the ELF, we know that there exists a polynomial $s(\log M, q)$ such that, with overwhelming probability over $\mathsf{ELF.Gen}(M, q)$,

$$\Pr_x[f(x) = z] \geq \frac{1}{s(\log M, q)}$$

for every element z in the image of f, where $\Pr_x$ denotes the probability over the randomness of x. Since $\log M$ is polynomial in λ, we conclude that our lossy distributed sampler is regular.

Programmability. We prove the property by means of a series of indistinguishable hybrids.

Hybrid 0. This hybrid corresponds to the programmability game in which $b = 0$. In particular, the distributed sampler message U_i received by the adversary as computed as follows.

1. $K \xleftarrow{\$} F.\mathsf{Gen}(\mathbb{1}^\lambda)$
2. $K_2^{(i)} \xleftarrow{\$} F_2.\mathsf{Gen}(\mathbb{1}^\lambda)$
3. $\mathsf{hk}_i \xleftarrow{\$} \mathsf{Hash.Gen}(\mathbb{1}^\lambda)$
4. $\mathsf{EP}_i \xleftarrow{\$} \mathsf{iO}(\mathbb{1}^\lambda, \mathsf{EProg}_{\mathsf{Ls}}[K_2^{(i)}, i])$ (see Fig. 30)
5. $\forall j \neq i: \quad \tau_e^j \xleftarrow{\$} \mathsf{NIZK.Trap}\big(\tau_e, (\mathsf{sid}, j)\big)$
6. $\mathsf{DP}_i \xleftarrow{\$} \mathsf{iO}(\mathbb{1}^\lambda, \mathsf{DProg}_{\mathsf{Ls}}[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j\neq i}, K, f])$ (see Fig. 31)
7. $\pi_i \xleftarrow{\$} \mathsf{NIZK.SimProve}\big(\tau_s, (\mathsf{sid}, i), (i, \mathsf{hk}_i, \mathsf{EP}_i)\big)$
8. $\pi'_i \xleftarrow{\$} \mathsf{NIZK'.SimProve}\big(\tau', (i, \mathsf{sid}, \mathsf{hk}_i, \mathsf{EP}_i, \mathsf{DP}_i, \pi_i, \sigma)\big)$
9. Output $U_i := (\mathsf{hk}_i, \mathsf{EP}_i, \mathsf{DP}_i, \pi_i, \pi'_i)$ and $\xi_e := K$.

During the sampling phase, after querying $U_j := (\mathsf{hk}_j, \mathsf{EP}_j, \mathsf{DP}_j, \pi_j, \pi'_j)$ for every $j \neq i$, the adversary is provided with a value R computed as follows:

1. $\forall j \in [n]: \quad b_j \leftarrow \mathsf{NIZK'.Verify}\big(\sigma', \pi'_j, (j, \mathsf{sid}, \mathsf{hk}_j, \mathsf{EP}_j, \mathsf{DP}_j, \pi_j, \sigma)\big)$
2. If there exists $j \in [n]$ such that $b_j = 0$, output $\perp$.
3. $z \leftarrow f\big((\mathsf{hk}_j, \mathsf{EP}_j)_{j\in[n]}\big)$
4. $s \leftarrow F(K, z)$
5. Output $\mathcal{D}(\mathbb{1}^\lambda; s)$.

Hybrid 1. In this hybrid, we modify the decryption program DP_i switching to an obfuscation of $\mathsf{DProg}_{\mathsf{Pr}}$ (see Fig. 38). In particular, the PRF key K hardcoded in the program will be punctured in the position z chosen by the adversary. Moreover, we hardcode into the program the value $R := \mathcal{D}(\mathbb{1}^\lambda; s)$ where $s = F(K, z)$. When the output of the ELF in the modified decryption program coincides with z, DP_i will directly input R in the partial decryption simulator. We formalise below the operations used for the generation of U_i.

1. $K \xleftarrow{\$} F.\mathsf{Gen}(\mathbb{1}^\lambda)$
2. $K^* \leftarrow F.\mathsf{Punct}(K, z)$
3. $s \leftarrow F(K, z)$
4. $R \leftarrow \mathcal{D}(\mathbb{1}^\lambda; s)$
5. $K_2^{(i)} \xleftarrow{\$} F_2.\mathsf{Gen}(\mathbb{1}^\lambda)$
6. $\mathsf{hk}_i \xleftarrow{\$} \mathsf{Hash.Gen}(\mathbb{1}^\lambda)$
7. $\mathsf{EP}_i \xleftarrow{\$} \mathsf{iO}(\mathbb{1}^\lambda, \mathsf{EProg}_{\mathsf{Ls}}[K_2^{(i)}, i])$ (see Fig. 30)
8. $\forall j \neq i: \quad \tau_e^j \xleftarrow{\$} \mathsf{NIZK.Trap}\big(\tau_e, (\mathsf{sid}, j)\big)$
9. $\mathsf{DP}_i \xleftarrow{\$} \mathsf{iO}(\mathbb{1}^\lambda, \mathsf{DProg}_{\mathsf{Pr}}[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j\neq i}, K^*, z, f, R])$ (see Fig. 38)
10. $\pi_i \xleftarrow{\$} \mathsf{NIZK.SimProve}\big(\tau_s, (\mathsf{sid}, i), (i, \mathsf{hk}_i, \mathsf{EP}_i)\big)$
11. $\pi'_i \xleftarrow{\$} \mathsf{NIZK'.SimProve}\big(\tau', (i, \mathsf{sid}, \mathsf{hk}_i, \mathsf{EP}_i, \mathsf{DP}_i, \pi_i, \sigma)\big)$
12. Output $U_i := (\mathsf{hk}_i, \mathsf{EP}_i, \mathsf{DP}_i, \pi_i, \pi'_i)$ and $\xi_e := K$.

Observe that this hybrid is indistinguishable from the previous one thanks to the security of iO.

Hybrid 2. In this hybrid, instead of generating R using the randomness output by F, we use an ideal sample. If, in the sampling phase, the adversary selects values $(U_j)_{j\neq i}$ such that $f\big((\mathsf{hk}_j, \mathsf{EP}_j)_{j\in[n]}\big) = z \neq \bot$ and, for every $j \neq i$,

$$\mathsf{NIZK}'.\mathsf{Verify}\big(\sigma', \pi'_j, (j, \mathsf{sid}, \mathsf{hk}_j, \mathsf{EP}_j, \mathsf{DP}_j, \pi_j, \sigma)\big) = 1,$$

the challenger immediately provides R to the adversary.

This hybrid is indistinguishable from the previous one by the security of the puncturable PRF F. We formalise below the operations used for the generation of U_i.

1. $K \xleftarrow{\$} F.\mathsf{Gen}(1\!\!1^\lambda)$
2. $K^* \leftarrow F.\mathsf{Punct}(K, z)$
3. $R \xleftarrow{\$} \mathcal{D}(1\!\!1^\lambda)$
4. $K_2^{(i)} \xleftarrow{\$} F_2.\mathsf{Gen}(1\!\!1^\lambda)$
5. $\mathsf{hk}_i \xleftarrow{\$} \mathsf{Hash}.\mathsf{Gen}(1\!\!1^\lambda)$
6. $\mathsf{EP}_i \xleftarrow{\$} \mathsf{iO}(1\!\!1^\lambda, \mathsf{EProg}_{\mathsf{Ls}}[K_2^{(i)}, i])$ (see Fig. 30)
7. $\forall j \neq i : \quad \tau_e^j \xleftarrow{\$} \mathsf{NIZK}.\mathsf{Trap}\big(\tau_e, (\mathsf{sid}, j)\big)$
8. $\mathsf{DP}_i \xleftarrow{\$} \mathsf{iO}(1\!\!1^\lambda, \mathsf{DProg}_{\mathsf{Pr}}[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j\neq i}, K^*, z, f, R])$ (see Fig. 38)
9. $\pi_i \xleftarrow{\$} \mathsf{NIZK}.\mathsf{SimProve}\big(\tau_s, (\mathsf{sid}, i), (i, \mathsf{hk}_i, \mathsf{EP}_i)\big)$
10. $\pi'_i \xleftarrow{\$} \mathsf{NIZK}'.\mathsf{SimProve}\big(\tau', (i, \mathsf{sid}, \mathsf{hk}_i, \mathsf{EP}_i, \mathsf{DP}_i, \pi_i, \sigma)\big)$
11. Output $U_i := (\mathsf{hk}_i, \mathsf{EP}_i, \mathsf{DP}_i, \pi_i, \pi'_i)$ and $\xi_e := K$.

We observe that the last hybrid is identical to the programmability game with $b = 1$.

□

C Proof of Hardness Preservation

Proof. Let $\mathsf{DS} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Sample}, \mathsf{LossySetup}, \mathsf{LossyGen}, \mathsf{Project}, \mathsf{Extract})$ be our n-party regular and programmable lossy distributed sampler for the distribution $\mathcal{D}(1\!\!1^\lambda)$. Let $\mathcal{A} \in \mathsf{AClass}$ be any PPT adversary such that, in the hardness-preserving game $\mathcal{G}$ in Fig. 20,

$$\Pr\left[\mathcal{G}^{\mathcal{A}}_{\mathsf{HP}}(1\!\!1^\lambda) = 1 \middle| b = 0\right] = \mathsf{nonegl}(\lambda).$$

Our goal is to prove that

$$\Pr\left[\mathcal{G}^{\mathcal{A}}_{\mathsf{HP}}(1\!\!1^\lambda) = 1 \middle| b = 1\right] = \mathsf{nonegl}(\lambda).$$

We define $\epsilon(\lambda) := \Pr\left[\mathcal{G}^{\mathcal{A}}_{\mathsf{HP}}(1\!\!1^\lambda) = 1 \middle| b = 0\right]$. Since $\epsilon(\lambda)$ is non-negligible, we know that there exists a polynomial $e(\lambda)$ such that for every $\overline{\lambda} \in \mathbb{N}$, there is a

$\lambda \geq \overline{\lambda}$ such that $\epsilon(\lambda) \geq 1/e(\lambda)$. Let $p(\lambda)$ be a polynomial upper-bounding twice the running times of $\mathcal{A}$.

We proceed by means of a Hybrid argument.

Hybrid 0. This stage corresponds to $\mathcal{G}^{\mathcal{A}}_{\mathsf{HP}}$. In particular, the challenger provides the adversary with a pair (crs, U_i) generated using the algorithms $\mathsf{Setup}(\mathbb{1}^\lambda)$ and $\mathsf{Gen}(\mathbb{1}^\lambda, \mathsf{sid}, i, \mathsf{crs})$. The sample given to $\mathcal{A}$ is instead computed using Sample.

Hybrid 1. In this hybrid, we change the distribution of crs, U_i and R. Specifically, we use the algorithms $\mathsf{LossySetup}\big(\mathbb{1}^\lambda, q(\lambda)\big)$, $\mathsf{LossyGen}(\mathbb{1}^\lambda, \mathsf{sid}, i, \zeta)$, $\mathsf{Project}\big(\zeta, (U_j)_{j\in[n]}, \mathsf{sid}\big)$ and $\mathsf{Extract}(\xi, z)$. The polynomial $q(\lambda)$ is chosen so that all adversaries running in time at most $p(\lambda)$ distinguish between the standard mode and the lossy mode parametrised by $q(\lambda)$ with advantage asymptotically smaller than $1/\big(2e(\lambda)\big)$. We denote the output of the adversary $\mathcal{A}$ after the interaction with the modified challenger by $\mathcal{G}^{\mathcal{A}}_{\mathsf{HP}_1}$.

Claim. In Hybrid 1, $\Pr\left[\mathcal{G}^{\mathcal{A}}_{\mathsf{HP}_1}(\mathbb{1}^\lambda) = 1\right] = \mathsf{nonegl}(\lambda)$.

Proof of the claim. Assume, by contradiction, that our claim is false. We construct an adversary that runs in time at most $p(\lambda)$ distinguishing between the standard mode and the lossy mode with advantage that is not asymptotically smaller than $1/\big(2e(\lambda)\big)$.

Our new adversary, denoted by $\mathcal{B}$, runs an internal copy of $\mathcal{A}$. The adversary $\mathcal{B}$ provides $\mathcal{A}$ with the value crs obtained from its challenger, after which, it obtains $i \in [n]$ and $\mathsf{sid} = (\mathsf{tag}, \mathsf{id}_{j_1}, \dots, \mathsf{id}_{j_n})$. The adversary $\mathcal{B}$ forwards $\mathsf{id}_{j_1}, \dots, \mathsf{id}_{j_n}$ to its challenger. Next, it issues a $\mathsf{NewSession}$ query with identity (sid, i). The answer U_i is forwarded to $\mathcal{A}$. When $\mathcal{A}$ replies with $(U_j)_{j\neq i}$, the adversary $\mathcal{B}$ queries $\big(\mathsf{Sample}, \mathsf{sid}, (U_j)_{j\neq i}\big)$ to its challenger and relays the result to $\mathcal{A}$. Finally, $\mathcal{B}$ outputs 1 if and only $\mathcal{A}$ outputs 1 and the distributed sampler output is not $\perp$.

The distinguishing advantage $\mathsf{Adv}_{\mathcal{B}}(\lambda)$ of $\mathcal{B}$ is $|\epsilon(\lambda) - \mathsf{negl}(\lambda)|$. For λ sufficiently big, we have that $\mathsf{Adv}_{\mathcal{B}}(\lambda)$ is greater than $\epsilon(\lambda) - 1/\big(4e(\lambda)\big)$. So, we conclude that for every $\overline{\lambda} \in \mathbb{N}$, there exists a $\lambda \geq \overline{\lambda}$ such that $\mathsf{Adv}_{\mathcal{B}}(\lambda) \geq 3/\big(4e(\lambda)\big)$. Since $\mathcal{B}$ at most in time $p(\lambda)$, we reached a contradiction. Notice that $\mathcal{B}$ is uniform if and only $\mathcal{A}$ is uniform. ■

Claim. Let E be the event in which $\mathcal{Z}(\zeta) = \mathsf{Project}\big(\zeta, (U_j)_{j\in[n]}, \mathsf{sid}\big)$. In Hybrid 1, we have

$$\Pr[\mathcal{G}^{\mathcal{A}}_{\mathsf{HP}_1}(\mathbb{1}^\lambda) = 1, E] = \mathsf{nonegl}(\lambda).$$

Proof of the claim.

Let V denote the event in which

$$\Pr_{\mathcal{Z}}\left[\mathcal{Z}(\zeta) = \mathsf{Project}\big(\zeta, (U_j)_{j\in[n]}, \mathsf{sid}\big)\right] < \frac{1}{s\big(\lambda, q(\lambda)\big)}$$

for some $(U_j)_{j\in[n]} \in \{0,1\}^*$ where the above probability is taken only over the randomness of $\mathcal{Z}$. By the regularity of the lossy distributed sampler $\Pr[V] =$

$\mathsf{negl}(\lambda)$. We conclude that

$$\begin{aligned}
&\Pr[\mathcal{G}^{\mathcal{A}}_{\mathsf{HP}_1}(\mathbb{1}^\lambda) = 1, E] \geq \\
\geq &\Pr[\mathcal{G}^{\mathcal{A}}_{\mathsf{HP}_1}(\mathbb{1}^\lambda) = 1, E, V] + \mathsf{negl}(\lambda) = \\
= &\Pr\left[E \middle| \mathcal{G}^{\mathcal{A}}_{\mathsf{HP}_1}(\mathbb{1}^\lambda) = 1, V\right] \cdot \Pr\left[\mathcal{G}^{\mathcal{A}}_{\mathsf{HP}_1}(\mathbb{1}^\lambda) = 1, V\right] + \mathsf{negl}(\lambda) \geq \\
\geq &\frac{1}{s\big(\lambda, q(\lambda)\big)} \cdot \Pr[\mathcal{G}^{\mathcal{A}}_{\mathsf{HP}_1}(\mathbb{1}^\lambda) = 1, V] + \mathsf{negl}(\lambda) \geq \\
\geq &\frac{1}{s\big(\lambda, q(\lambda)\big)} \cdot \Pr[\mathcal{G}^{\mathcal{A}}_{\mathsf{HP}_1}(\mathbb{1}^\lambda) = 1] + 2 \cdot \mathsf{negl}(\lambda).
\end{aligned}$$

We conclude the proof of the claim by observing that $\Pr[\mathcal{G}^{\mathcal{A}}_{\mathsf{HP}_1}(\mathbb{1}^\lambda) = 1]$ is non-negligible by Claim C. ■

Hybrid 2. In this hybrid, we generate U_i and ξ using $\mathsf{ProgGen}(\mathbb{1}^\lambda, \mathsf{sid}, i, z, R, \zeta)$ where $R \xleftarrow{\$} \mathcal{D}(\mathbb{1}^\lambda)$ and $z \xleftarrow{\$} \mathcal{Z}(\zeta)$. If the adversary selects $(U_j)_{j \neq i}$ such that $\mathsf{Project}\big(\zeta, (U_j)_{j\in[n]}, \mathsf{sid}\big) = z$ and $z \neq \bot$, we provide the adversary with R. If instead $\mathsf{Project}\big(\zeta, (U_j)_{j\in[n]}, \mathsf{sid}\big) = \bot$, we provide the adversary with $\bot$. The rest remains as in the previous hybrid. We denote the output of the adversary $\mathcal{A}$ after the interaction with the modified challenger by $\mathcal{G}^{\mathcal{A}}_{\mathsf{HP}_2}$.

Claim. In the new game $\mathcal{G}^{\mathcal{A}}_{\mathsf{HP}_2}$, we have

$$\Pr[\mathcal{G}^{\mathcal{A}}_{\mathsf{HP}_2}(\mathbb{1}^\lambda) = 1, E] = \mathsf{nonegl}(\lambda).$$

Proof of the claim. Suppose that our claim is false. Then, we can find a PPT adversary $\mathcal{B}$ that breaks the programmability of the lossy distributed sampler. The adversary $\mathcal{B}$ provides the CRS it receives from its challenger to an internal copy of $\mathcal{A}$. Then, it samples $z \xleftarrow{\$} \mathcal{Z}(\zeta)$. Notice that ζ is given to $\mathcal{B}$ by its challenger. When $\mathcal{A}$ selects sid and $i \in [n]$, $\mathcal{B}$ sends sid, i, z to its challenger. It then proceeds by relaying all the communications between $\mathcal{A}$ and its challenger. At the end of its execution, $\mathcal{B}$ outputs 1 if and only if $\mathcal{A}$ outputs 1, the distributed sampler output is not $\bot$ and $\mathsf{Project}\big(\zeta, (U_j)_{j\in[n]}, \mathsf{sid}\big) = z$.

Notice that the advantage of $\mathcal{B}$ is

$$\left|\Pr[\mathcal{G}^{\mathcal{A}}_{\mathsf{HP}_1}(\mathbb{1}^\lambda) = 1, E] - \Pr[\mathcal{G}^{\mathcal{A}}_{\mathsf{HP}_2}(\mathbb{1}^\lambda) = 1, E]\right| = \mathsf{nonegl}(\lambda) - \mathsf{negl}(\lambda).$$

Observe also that $\mathcal{B}$ is uniform if and only $\mathcal{A}$ is uniform. We reached a contradiction. ■

Hybrid 3. In this hybrid, we modify $\mathcal{G}^{\mathcal{A}}_{\mathsf{HP}_2}$. Instead of providing $\mathcal{A}$ with $\mathsf{Extract}\Big(\xi, \mathsf{Project}\big(\zeta, (U_j)_{j\in[n]}, \mathsf{sid}\big)\Big)$, we now provide it with the value R hidden in U_i. We call the resulting game $\mathcal{G}^{\mathcal{A}}_{\mathsf{HP}_3}$.

Claim. In the new game $\mathcal{G}^{\mathcal{A}}_{\mathsf{HP}_3}$, we have $\Pr[\mathcal{G}^{\mathcal{A}}_{\mathsf{HP}_3}(\mathbb{1}^\lambda) = 1] = \mathsf{nonegl}(\lambda)$.

Proof of the claim. We prove that $\Pr[\mathcal{G}^{\mathcal{A}}_{\mathsf{HP}_3}(\mathbb{1}^\lambda) = 1, E] = \mathsf{nonegl}(\lambda)$. The result follows from the fact that $\Pr[\mathcal{G}^{\mathcal{A}}_{\mathsf{HP}_3}(\mathbb{1}^\lambda) = 1] \geq \Pr[\mathcal{G}^{\mathcal{A}}_{\mathsf{HP}_3}(\mathbb{1}^\lambda) = 1, E]$.

We notice that in both $\mathcal{G}^{\mathcal{A}}_{\mathsf{HP}_2}$ and $\mathcal{G}^{\mathcal{A}}_{\mathsf{HP}_3}$, when E occurs, the adversary is always provided with either R or $\perp$. In the second case, both $\mathcal{G}^{\mathcal{A}}_{\mathsf{HP}_2}(\mathbb{1}^\lambda) = 0$ and $\mathcal{G}^{\mathcal{A}}_{\mathsf{HP}_3}(\mathbb{1}^\lambda) = 0$. We conclude that, by Claim C,

$$\Pr[\mathcal{G}^{\mathcal{A}}_{\mathsf{HP}_3}(\mathbb{1}^\lambda) = 1, E] = \Pr[\mathcal{G}^{\mathcal{A}}_{\mathsf{HP}_2}(\mathbb{1}^\lambda) = 1, E] = \mathsf{nonegl}(\lambda).$$

■

Hybrid 3 corresponds to the ideal world execution of the hardness-preserving distributed sampler. In particular, $\mathsf{SimSetup}_{\mathcal{A}}$ just performs the same operations as $\mathsf{LossySetup}$. The simulator $\mathsf{SimGen}_{\mathcal{A}}(\mathbb{1}^\lambda, \mathsf{sid}, i, \zeta, R)$ instead outputs the message U_i generate by $\mathsf{ProgGen}(\mathbb{1}^\lambda, \mathsf{sid}, i, z, R, \zeta)$ where $z \xleftarrow{\$} \mathcal{Z}(\zeta)$. Notice that the polynomial $q(\lambda)$ used for the ELF depends on the running time of $\mathcal{A}$. Claim C proves that

$$\Pr\left[\mathcal{G}^{\mathcal{A}}_{\mathsf{HP}}(\mathbb{1}^\lambda) = 1 \middle| b = 1\right] = \mathsf{nonegl}(\lambda).$$

□

D Proof of Indistinguishability Preservation

Proof. Let $\mathcal{A}$ be any PPT adversary in AClass that distinguishes between $\mathcal{G}'_0$ and $\mathcal{G}'_1$ with non-negligible advantage $\epsilon(\lambda)$. In particular, we know that there exists a polynomial $e(\lambda)$ such that, for every $\overline{\lambda} \in \mathbb{N}$, there exists a $\lambda \geq \overline{\lambda}$ such that $\epsilon(\lambda) \geq 1/e(\lambda)$. We proceed by means of a hybrid argument starting from $\mathcal{G}'_0$. Let i be the index of a honest party. Let $p(\lambda)$ be a polynomial upper-bounding the running time of $\mathcal{A}$. Let $p'(\lambda)$ be a polynomial upper-bounding the running time of the challengers in $\mathcal{G}'_0$ and $\mathcal{G}'_1$ when the adversary runs in time at most $p(\lambda)$. We select the polynomial $q(\lambda)$ so that every adversary running in time at most $p(\lambda) + p'(\lambda)$ distinguishes between the standard mode and the lossy mode of both the ELF and the distributed sampler with advantage definitively smaller than $1/(4e(\lambda))$. Notice that by Theorem 16, such polynomial $q(\lambda)$ exists.

Hybrid 0. This hybrid corresponds to $\mathcal{G}'_0$

Hybrid 1. In this hybrid, the challenger witched the distributed sampler to lossy mode. Specifically, it generates the distributed sampler CRS crs using $\mathsf{LossySetup}\big(\mathbb{1}^\lambda, q(\lambda)\big)$. Furthermore, in every $\mathsf{NewSession}$ query, it generates the last distributed sampler message sent by a honest party using the lossy mode of the distributed sampler, i.e.,

$$U_i \xleftarrow{\$} \mathsf{LossyGen}(\mathbb{1}^\lambda, \mathsf{sid}, i, \zeta).$$

Finally, when all the distributed sampler messages have been exchanged, the challenger computes the output R using $\mathsf{Project}$ and $\mathsf{Extract}$ instead of Sample. The rest remains exactly as in $\mathcal{G}'_0$.

Notice that, by the first property of lossy distributed samplers, the distinguishable advantage of $\mathcal{A}$ between Hybrid 0 and Hybrid 1 is asymptotically smaller than $1/(4e(\lambda))$. The reduction is pretty straightforward. The new adversary $\mathcal{B}$ receives the CRS crs from the lossy distributed sampler challenger. It uses the latter to simulate $\mathcal{G}_0'$ to an internal copy of $\mathcal{A}$. When $\mathcal{A}$ sends a new session identity sid, $\mathcal{B}$ performs the same operations as the challenger in $\mathcal{G}_0'$. Things change when the last honest party sends its distributed sampler message. Let id_{j_i} be the identity of the corresponding party. The adversary $\mathcal{B}$ queries $(\mathsf{NewSession}, \mathsf{sid}, i)$ to its challenger and relays the answer to $\mathcal{A}$. Then, when all the distributed sampler messages $(U_j)_{j\in[n]}$ have been exchanged, $\mathcal{B}$ queries $\big(\mathsf{Sample}, \mathsf{sid}, (U_j)_{j\neq i}\big)$ to its challenger and provides the answer to the copy of Ch_0. The new adversary $\mathcal{B}$ outputs the same value as $\mathcal{A}$. We observe that if $\mathcal{A}$ is uniform, $\mathcal{B}$ is uniform too. Furthermore, the running time of $\mathcal{B}$ is at most $p(\lambda) + p'(\lambda)$. By the first property of lossy distributed samplers, the advantage of $\mathcal{B}$ is asymptotically smaller than $1/(4e(\lambda))$.

Hybrid 2. In this hybrid, the challenger uses Ch_1 instead of Ch_0 in every $\mathsf{NewSession}$ query. Notice that Ch_1 is just provided with a sample R, but not with any trapdoor T.

Claim. No PPT adversary $\mathcal{A}$ can distinguish between Hybrid 1 and Hybrid 2.

Proof of the claim. Let $M(\lambda)$ be a polynomial upper-bound on the number of $\mathsf{NewSession}$ queries issued by the adversary $\mathcal{A}$. For every $\iota \in [M] \cup \{0\}$, we define Hybrid' ι in which the first ι $\mathsf{NewSession}$ queries are dealt using Ch_1, whereas the rest are dealt using Ch_0. We prove that, for every $\iota \in [M]$, no PPT adversary can distinguish between Hybrid' $\iota - 1$ and Hybrid' ι.

We do this by means of a reduction to the chosen-sample indistinguishability of $\mathcal{G}_0$ and $\mathcal{G}_1$. In the reduction, we build a new adversary $\mathcal{B}$ having a copy of $\mathcal{A}$. The adversary $\mathcal{B}$ starts its execution by producing a distributed sampler CRS crs using $\mathsf{LossySetup}$. In the process, it obtains also ζ. In the first $\iota - 1$ $\mathsf{NewSession}$ queries, $\mathcal{B}$ simulates the game in Hybrid 1 using Ch_1 as challenger. Starting from the $(\iota + 1)$-th query, $\mathcal{B}$ uses instead Ch_0. For the ι-th session, $\mathcal{B}$ sends the corresponding auxiliary information aux and the set of honest parties $H' := \{l \in [n] | j_l \in H\}$ to its challenger. It then relays the messages between its challenger and $\mathcal{A}$. In all the sessions, including the ι-th one, $\mathcal{B}$ generates the distributed sampler messages as in Hybrid 1. In particular, the last honest distributed sampler message sent in every session is produced using $\mathsf{LossyGen}$ and ζ. Furthermore, the output of the distributed sampler is computed using $\mathsf{Project}$ and $\mathsf{Extract}$. In the ι-th session, $\mathcal{B}$ gives the output of $\mathsf{Extract}$ to its challenger.

We have just proven that Hybrid' $\iota - 1$ and Hybrid' ι are indistinguishable for every $\iota \in [M]$. We conclude that Hybrid' 0 and Hybrid' M are indistinguishable too. The latter are identical to Hybrid 1 and Hybrid 2 respectively. That ends the proof of the claim. ■

Hybrid 3. For any session of identity $\mathsf{sid} = (\mathsf{tag}, \mathsf{id}_{j_1}, \dots, \mathsf{id}_{j_n})$, let j_i be the index of the last honest party sending a distributed sampler message. In this hybrid, the challenger generates the decryption program DP_i in U_i by obfuscating the program $\mathsf{DProg}_{\mathsf{IP}}$ (see Fig. 41) instead of $\mathsf{DProg}_{\mathsf{Ls}}$ (see Fig. 31). In $\mathsf{DProg}_{\mathsf{IP}}$ we hardcode the auxiliary information aux' given by Ch_1. Notice that we now generate the outputs of the distributed sampler using the trapdoored distribution $\mathcal{D}'(\mathbb{1}^\lambda, \mathsf{aux}')$.

After computing the output of the distributed sampler $\hat{R}$, the challenger provides $\mathcal{G}_1$ with a trapdoor $\hat{T}$. The latter is retrieved by rerunning the computations of DP_i in clear. Specifically, we perform the following operations

1. $z \leftarrow \mathsf{Project}\big(\zeta, (U_j)_{j\in[n]}, \mathsf{sid}\big)$
2. $s \leftarrow F(\xi, z)$
3. $(\hat{R}, \hat{T}) \leftarrow \mathcal{D}'(\mathbb{1}^\lambda, \mathsf{aux}'; s)$

We recall that ξ is computed by $\mathsf{LossyGen}$ together with U_i and consists of the PPRF key K hardcoded in DP_i.

Claim. Hybrid 3 is computationally indistinguishable from Hybrid 2.

Proof of the claim. Let $N(\lambda)$ be a polynomial upper-bound on the number of $\mathsf{NewSession}$ queries issued by the adversary $\mathcal{A}$. For every $\iota \in [N] \cup \{0\}$, we define Hybrid' ι in which the first ι $\mathsf{NewSession}$ queries are answered as in Hybrid 3. The remaining sessions are answered as in Hybrid 2. Notice that Hybrid' 0 is identical to Hybrid 2. Similarly, Hybrid' N is identical to Hybrid 3. We show that, for every $i \in [N]$, Hybrid' ι and Hybrid' $\iota - 1$ are computationally indistinguishable. That will immediately imply our claim.

We prove that Hybrid' ι and Hybrid' $\iota - 1$ are indistinguishable by means of a sequence of indistinguishable hybrids.

Hybrid" 0. In this hybrid, we answer the first $\iota - 1$ $\mathsf{NewSession}$ queries as in Hybrid" 3. Starting from the $(\iota + 1)$-th query, we instead answer as in Hybrid" 2. We deviate from the usual behaviour in the ι-th query. We sample $x \xleftarrow{\$} [M]$ and we compute $\widehat{z} \leftarrow f(x)$. Then, if $\widehat{z} \neq f\big((\mathsf{hk}_j, \mathsf{EP}_j)_{j\in[n]}\big)$, where $(\mathsf{hk}_j, \mathsf{EP}_j)_{j\in[n]}$ are the hash keys and encryption programs exchanged in the ι-th session, we rewind the adversary and we retry. Observe that, with overwhelming probability, we succeed within $t(\lambda)$ tries for some polynomial t. This hybrid is perfectly indistinguishable from Hybrid' $\iota - 1$.

Hybrid" 1. In this hybrid, we behave as in Hybrid" 0 with minor changes in the answer to the ι-th $\mathsf{NewSession}$ query. Let id_{j_i} be the identity of the last honest party sending a distributed sampler message in the ι-th session. We generate the decryption program DP_i by obfuscating $\mathsf{DProg}^0_{\mathsf{Ls}}$ (see Fig. 61): after receiving the input $(\mathsf{hk}_j, \mathsf{EP}_j, \pi_j)_{j\neq i}$, $\mathsf{DProg}^0_{\mathsf{Ls}}$ immediately checks if $f\big((\mathsf{hk}_j, \mathsf{EP}_j)_{j\in[n]}\big) = \hat{z}$. If that is the case, instead of inputting $\hat{R}$ into the partial decryption simulator, the program inputs $R := \mathcal{D}(\mathbb{1}^\lambda; s)$ where $s = F(K, \hat{z})$. Such value R is hardcoded into $\mathsf{DProg}^0_{\mathsf{Ls}}$. If instead $f\big((\mathsf{hk}_j, \mathsf{EP}_j)_{j\in[n]}\big) \neq \hat{z}$, the program computes the output using K^* instead of K, where K^* denotes the puncturing of K in position $\hat{z}$. Observe that the input-output behaviour of the program remains the same as in the previous hybrid. We conclude that Hybrid" 0 and Hybrid" 1 are indistinguishable due to the security of iO.

$\mathsf{DProg}^0_{\mathsf{Ls}}[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j \neq i}, K^*, f, \hat{z}, R]$

Hard-coded. The index i of the party, the session identity sid, a PPRF key $K_2^{(i)}$, the encryption program EP_i, the hash key hk_i, the extractable NIZK CRS σ and the extraction trapdoors $(\tau_e^j)_{j \neq i}$, the punctured PRF key K^*, the ELF f, the value $\hat{z}$, the sample R.
Input. Set of $n-1$ tuples $(\mathsf{hk}_j, \mathsf{EP}_j, \pi_j)_{j \neq i}$.

```
1. If $f\big((\mathsf{hk}_j, \mathsf{EP}_j)_{j \in [n]}\big) \neq \hat{z}$, output
   $d_i \leftarrow \mathsf{DProg}_{\mathsf{Ls}}[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j \neq i}, K^*, f]\big((\mathsf{hk}_j, \mathsf{EP}_j, \pi_j)_{j \neq i}\big)$ (see Fig. 31)
2. If $f\big((\mathsf{hk}_j, \mathsf{EP}_j)_{j \in [n]}\big) = \hat{z}$, perform the following operations:
   (a) $\forall j \neq i: \quad b_j \leftarrow \mathsf{NIZK.Verify}\big(\sigma, (\mathsf{sid}, j), \pi_j, (\mathsf{hk}_j, \mathsf{EP}_j)\big)$
   (b) $\forall j \neq i: \quad \big(K_1^{(j)}, K_2^{(j)}\big) \leftarrow \mathsf{NIZK.Extract}\big(\tau_e^j, \pi_j, (j, \mathsf{hk}_j, \mathsf{EP}_j)\big)^a$
   (c) If $\exists j \neq i$ such that $b_j = 0$ or $\big(K_1^{(j)}, K_2^{(j)}\big) = \perp$, output $\perp$
   (d) $\forall j \in [n]: \quad y_j \leftarrow \mathsf{Hash}\big(\mathsf{hk}_j, (\mathsf{hk}_l, \mathsf{EP}_l)_{l \neq j}\big)$
   (e) $\forall j \neq i: \quad s_j \leftarrow F_1\big(K_1^{(j)}, y_j\big)$
   (f) $\forall j \in [n]: \quad (r_j, r'_j, r''_j, \eta_j, \eta'_j) \leftarrow F_2\big(K_2^{(j)}, y_j\big)$
   (g) $(\phi, \mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{mkFHE.Sim}_1(1\!\!1^\lambda, i; r''_i)$
   (h) $d_i \leftarrow \mathsf{mkFHE.Sim}_2\Big(\phi, \tilde{\mathcal{D}}, R, (s_j, r_j, r'_j)_{j \neq i}; \eta'_i\Big)$ (see bottom of Fig. 29)
   (i) Output $d_i$
```

Fig. 60. The unobfuscated decryption program for Hybrid" 1

Hybrid" 2. In this hybrid, in the ι-th session, instead of generating the sample R hardcoded into DP_i using the randomness produced by $F(K, \hat{z})$, the challenger simply samples $R \xleftarrow{\$} \mathcal{D}(1\!\!1^\lambda)$. When all the distributed sampler messages have been exchanged, the challenger verifies the NIZKs. If any check fails or $f\big((\mathsf{hk}_j, \mathsf{EP}_j)_{j \in [n]}\big) \neq \hat{z}$, the challenger behaves as before. Otherwise, it directly provides R to Ch_1. Since we provide the adversary only with K^*, this hybrid is indistinguishable from the previous one by the security of the puncturable PRF F.

Hybrid" 3. In this hybrid, in the ι-th session, the challenger generates the sample R hardcoded in DP_i using $(R, T) \xleftarrow{\$} \mathcal{D}'(1\!\!1^\lambda, \mathsf{aux}')$. This hybrid is indistinguishable from the previous one since $\mathcal{D}'$ is a trapdoored distribution for $\mathcal{D}$.

Hybrid" 4. In this hybrid, in the ι-th session, if all the NIZKs in the distributed sampler messages verify and $f\big((\mathsf{hk}_j, \mathsf{EP}_j)_{j \in [n]}\big) = \hat{z}$, the challenger provides Ch_1 with the trapdoor T produced by $\mathcal{D}'$ along with R. This hybrid is indistinguishable from the previous one by the trapdoor security of $\mathcal{G}_1$.

In the reduction, we build an adversary $\mathcal{B}$ holding a copy of $\mathcal{A}$. Upon activation, $\mathcal{B}$ simulates the $\mathcal{G}'_0$ as in Hybrid" 3 to $\mathcal{A}$. It behaves differently in the ι-th session. Let aux be corresponding auxiliary input and let $\mathsf{sid} = (\mathsf{tag}, \mathsf{id}_{j_1}, \ldots, \mathsf{id}_{j_n})$

be the corresponding session identity. Let id_{j_i}, be the identity of the last honest party sending a distributed sampler message. The adversary $\mathcal{B}$ provides its challenger with aux and $H' := \{l \in [n] | j_l \in H\}$. Then, it keeps relaying the messages between its challenger and $\mathcal{A}$. The distributed sampler messages in the ι-th session are produced by $\mathcal{B}$ as in Hybrid" 3, except for the fact that $\mathcal{B}$ uses the sample R provided by its challenger when it is time to generate DP_i. When all the distributed sampler messages have been sent, $\mathcal{B}$ checks whether the NIZKs verify and $f\big((\mathsf{hk}_j, \mathsf{EP}_j)_{j\in[n]}\big) = \hat{z}$. If that is the case, $\mathcal{B}$ keeps relaying the messages between its challenger and $\mathcal{A}$ and outputs the same value as $\mathcal{A}$. In the other cases, $\mathcal{B}$ simply outputs a random bit. Observe that the advantage of $\mathcal{B}$ against the trapdoor security of $\mathcal{G}_1$ is the same as the advantage of $\mathcal{A}$ in distinguishing between Hybrid" 3 and 4.

Hybrid" 5. In this hybrid, in the ι-th session, we generate the sample R hardcoded in DP_i using $(R, T) \xleftarrow{\$} \mathcal{D}'(\mathbb{1}^\lambda, \mathsf{aux}'; s)$ where $s = F(K, \hat{z})$. All the rest remains as in the previous hybrid. Since we provide the adversary only with K^*, this hybrid is indistinguishable from the previous one by the security of the puncturable PRF F.

We now proceed with a sequence of $q(\lambda)$ hybrids, where $q(\lambda)$ is the polynomial given as input to $\mathsf{LossySetup}$ (the total number of hybrids is polynomial). Let γ denote the ω-th element in the image of f that differs from $\hat{z}$, if we order the latter according to the lexicographical order. Notice that since the ELF is regular, it is also strongly efficiently enumerable [Zha16], so, given f, we can efficiently compute γ. Throughout the proof, we assume that f has an image with at most $q(\lambda)$ elements and that the challenger successfully retrieves the whole image of f. This is enough to prove our claim as these events occur with overwhelming probability.

Hybrid" ω.0. We behave as in Hybrid" 5, except in the ι-th $\mathsf{NewSession}$ query. Let id_{j_i} be the identity of the last honest party sending a distributed sampler message in the ι-th session. We generate the decryption program DP_i by obfuscating $\mathsf{DProg}^1_{\mathsf{Ls}}$ (see Fig. 61): after receiving the input $(\mathsf{hk}_j, \mathsf{EP}_j, \pi_j)_{j\neq i}$, $\mathsf{DProg}^1_{\mathsf{Ls}}$ immediately checks if $f\big((\mathsf{hk}_j, \mathsf{EP}_j)_{j\in[n]}\big) <_{\mathsf{lex}} \gamma$ or $f\big((\mathsf{hk}_j, \mathsf{EP}_j)_{j\in[n]}\big) = \hat{z}$. If that is the case, it behaves as $\mathsf{DProg}_{\mathsf{IP}}$ (see Fig. 41), otherwise, it performs the same operations as $\mathsf{DProg}_{\mathsf{Ls}}$ (see Fig. 31).

We observe that if $\omega = 1$, this hybrid is indistinguishable from Hybrid" 5 by the security of iO. Indeed, x is the minimum in the image of f, so $f\big((\mathsf{hk}_j, \mathsf{EP}_j)_{j\in[n]}\big)$ will always be greater or equal to x. In other words, DP_i will always behave as $\mathsf{DProg}^0_{\mathsf{Ls}}$. If instead $\omega > 1$, this hybrid is identical to the previous one, i.e. Hybrid" $(\omega - 1)$.5.

Hybrid" ω.1. In this hybrid, we change the decryption program DP_i of the last honest party sending a distributed sampler message in the ι-th session. Specifically, instead of obfuscating $\mathsf{DProg}^1_{\mathsf{Ls}}$, we obfuscate $\mathsf{DProg}^2_{\mathsf{Ls}}$ (see Fig. 62). In the latter, the PRF key K is punctured in position γ, we denote it by K^*. After receiving the input $(\mathsf{hk}_j, \mathsf{EP}_j, \pi_j)_{j\neq i}$, $\mathsf{DProg}^2_{\mathsf{Ls}}$ immediately checks if $f\big((\mathsf{hk}_j, \mathsf{EP}_j)_{j\in[n]}\big) <_{\mathsf{lex}} \gamma$ or $f\big((\mathsf{hk}_j, \mathsf{EP}_j)_{j\in[n]}\big) = \hat{z}$. If that is the case, it behaves as $\mathsf{DProg}_{\mathsf{IP}}$ (see Fig. 41). Otherwise, $\mathsf{DProg}^2_{\mathsf{Ls}}$ performs the same operations as

$\mathsf{DProg}^1_{\mathsf{Ls}}[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j \neq i}, K, f, \hat{z}, \mathsf{aux}', \gamma]$

Hard-coded. The index i of the party, the session identity sid, a PPRF key $K_2^{(i)}$, the encryption program EP_i, the hash key hk_i, the extractable NIZK CRS σ and the extraction trapdoors $(\tau_e^j)_{j \neq i}$, the PPRF key K, the ELF f, the value $\hat{z}$, the auxiliary information aux', the hybrid index γ.
Input. Set of $n-1$ tuples $(\mathsf{hk}_j, \mathsf{EP}_j, \pi_j)_{j \neq i}$.

1. If $f\big((\mathsf{hk}_j, \mathsf{EP}_j)_{j \in [n]}\big) <_{\mathsf{lex}} \gamma$ or $f\big((\mathsf{hk}_j, \mathsf{EP}_j)_{j \in [n]}\big) = \hat{z}$, output $d_i \leftarrow \mathsf{DProg}_{\mathsf{IP}}[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j \neq i}, K, f, \mathsf{aux}']\big((\mathsf{hk}_j, \mathsf{EP}_j, \pi_j)_{j \neq i}\big)$ (see Fig. 41)
2. Otherwise, output $d_i \leftarrow \mathsf{DProg}_{\mathsf{Ls}}[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j \neq i}, K, f]\big((\mathsf{hk}_j, \mathsf{EP}_j, \pi_j)_{j \neq i}\big)$ (see Fig. 31)

Fig. 61. The unobfuscated decryption program for Hybrid” $\omega.0$

$\mathsf{DProg}_{\mathsf{Ls}}$ (see Fig. 31) with only one exception: when $f\big((\mathsf{hk}_j, \mathsf{EP}_j)_{j \in [n]}\big) = \gamma$, instead of inputting $\hat{R}$ into the partial decryption simulator, the program inputs $R := \mathcal{D}(1\!\!1^\lambda; s)$ where $s = F(K, \gamma)$. Such value R is hardcoded into $\mathsf{DProg}^2_{\mathsf{Ls}}$.

All the rest remains as in the previous hybrid. Since the input-output behaviour of DP_i has not changed, this hybrid is indistinguishable from the previous one by the security of iO.

Hybrid” ω.2. In this hybrid, in the ι-th session, instead of generating the sample R hardcoded into DP_i using the randomness produced by $F(K, \gamma)$, the challenger simply samples $R \xleftarrow{\$} \mathcal{D}(1\!\!1^\lambda)$. Since we provide the adversary only with K^*, this hybrid is indistinguishable from the previous one by the security of the puncturable PRF F.

Hybrid” ω.3. In this hybrid, in the ι-th session, the challenger generates the sample R hardcoded in DP_i using $(R, T) \xleftarrow{\$} \mathcal{D}'(1\!\!1^\lambda, \mathsf{aux}')$. This hybrid is indistinguishable from the previous one since $\mathcal{D}'$ is a trapdoored distribution for $\mathcal{D}$.

Hybrid” ω.4. In this hybrid, in the ι-th session, we generate the sample R hardcoded in DP_i using $(R, T) \xleftarrow{\$} \mathcal{D}'(1\!\!1^\lambda, \mathsf{aux}'; s)$ where $s = F(K, \gamma)$. All the rest remains as in the previous hybrid. Since we provide the adversary only with K^*, this hybrid is indistinguishable from the previous one by the security of the puncturable PRF F.

Hybrid” ω.5. In this hybrid, in the ι-th session, the challenger generates the decryption program DP_i by obfuscating $\mathsf{DProg}^1_{\mathsf{Ls}}$ (see Fig. 61), however, instead of hardcoding γ, it will hardcode the next value in the image of f that differs from $\hat{z}$[21]. In other words, DP_i will behave as $\mathsf{DProg}_{\mathsf{IP}}$ (see Fig. 41) whenever

[21] If γ is already the maximum in the image of f, we augment the latter with an imaginary element that is strictly greater than all other values and we hardcode it into $\mathsf{DProg}^1_{\mathsf{Ls}}$.

$\mathsf{DProg}^2_{\mathsf{Ls}}[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j\neq i}, K^*, f, \hat{z}, \mathsf{aux}', \gamma, R]$

Hard-coded. The index i of the party, the session identity sid, a PPRF key $K_2^{(i)}$, the encryption program EP_i, the hash key hk_i, the extractable NIZK CRS σ and the extraction trapdoors $(\tau_e^j)_{j\neq i}$, the punctured PRF key K^*, the ELF f, the value $\hat{z}$, the auxiliary information aux', the hybrid index γ, the sample R.
Input. Set of $n-1$ tuples $(\mathsf{hk}_j, \mathsf{EP}_j, \pi_j)_{j\neq i}$.

```
1. If $f\big((\mathsf{hk}_j, \mathsf{EP}_j)_{j\in[n]}\big) <_{\mathsf{lex}} \gamma$ or $f\big((\mathsf{hk}_j, \mathsf{EP}_j)_{j\in[n]}\big) = \hat{z}$, output
   $d_i \leftarrow \mathsf{DProg}_{\mathsf{IP}}[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j\neq i}, K^*, f, \mathsf{aux}']\big((\mathsf{hk}_j, \mathsf{EP}_j, \pi_j)_{j\neq i}\big)$ (see Fig. 41)
2. If $f\big((\mathsf{hk}_j, \mathsf{EP}_j)_{j\in[n]}\big) = \gamma$, perform the following operations:
   (a) $\forall j \neq i: \quad b_j \leftarrow \mathsf{NIZK.Verify}\big(\sigma, (\mathsf{sid}, j), \pi_j, (\mathsf{hk}_j, \mathsf{EP}_j)\big)$
   (b) $\forall j \neq i: \quad \big(K_1^{(j)}, K_2^{(j)}\big) \leftarrow \mathsf{NIZK.Extract}\big(\tau_e^j, \pi_j, (j, \mathsf{hk}_j, \mathsf{EP}_j)\big)^a$
   (c) If $\exists j \neq i$ such that $b_j = 0$ or $\big(K_1^{(j)}, K_2^{(j)}\big) = \perp$, output $\perp$
   (d) $\forall j \in [n]: \quad y_j \leftarrow \mathsf{Hash}\big(\mathsf{hk}_j, (\mathsf{hk}_l, \mathsf{EP}_l)_{l\neq j}\big)$
   (e) $\forall j \neq i: \quad s_j \leftarrow F_1\big(K_1^{(j)}, y_j\big)$
   (f) $\forall j \in [n]: \quad (r_j, r'_j, r''_j, \eta_j, \eta'_j) \leftarrow F_2\big(K_2^{(j)}, y_j\big)$
   (g) $(\phi, \mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{mkFHE.Sim}_1(1\!\!1^\lambda, i; r''_i)$
   (h) $d_i \leftarrow \mathsf{mkFHE.Sim}_2\Big(\phi, \tilde{\mathcal{D}}, R, (s_j, r_j, r'_j)_{j\neq i}; \eta'_i\Big)$ (see bottom of Fig. 29)
   (i) Output $d_i$
3. Otherwise, output
   $d_i \leftarrow \mathsf{DProg}_{\mathsf{Ls}}[i, \mathsf{sid}, K_2^{(i)}, \mathsf{EP}_i, \mathsf{hk}_i, \sigma, (\tau_e^j)_{j\neq i}, K^*, f]\big((\mathsf{hk}_j, \mathsf{EP}_j, \pi_j)_{j\neq i}\big)$ (see Fig. 31)
```

Fig. 62. The unobfuscated decryption program for Hybrid" $\omega.1$

$f\big((\mathsf{hk}_j, \mathsf{EP}_j)_{j\in[n]}\big) \leq_{\mathsf{lex}} \gamma$ or $f\big((\mathsf{hk}_j, \mathsf{EP}_j)_{j\in[n]}\big) = \hat{z}$. In the other cases, it will behave as $\mathsf{DProg}_{\mathsf{Ls}}$ (see Fig. 31). Notice that the input-output behaviour of DP_i has not changed. We conclude that this hybrid is indistinguishable from the previous one by the security of iO.

When $\omega = q(\lambda)$, the last hybrid is indistinguishable from Hybrid 3 by the security of iO. Indeed, DP_i always behaves as $\mathsf{DProg}_{\mathsf{IP}}$ as there are no elements in the image of f such that $f\big((\mathsf{hk}_j, \mathsf{EP}_j)_{j\in[n]}\big) >_{\mathsf{lex}} \gamma$.

This ends the proof of the claim. ■

Hybrid 4. In this hybrid, we switch the ELF to injective mode. Observe that this stage corresponds to game $\mathcal{G}'_1$. Observe that the distinguishability advantage of $\mathcal{A}$ against Hybrid 3 and Hybrid 4 is at most $1/(4e(\lambda))$.

We conclude that the distinguishability advantage of $\mathcal{A}$ between $\mathcal{G}'_0$ and $\mathcal{G}'_1$ is at most $1/(2e(\lambda)) + \mathsf{negl}(\lambda)$. The latter is asymptotically smaller than $1/e(\lambda)$. We reached a contradiction, so no PPT adversary can distinguish between $\mathcal{G}'_0$ and $\mathcal{G}'_1$. □